GUIDE PRATIQUE

CP/M PAS A PAS

ALAIN PINAUD

EDITIONS DU P.S.I.

CP/M
PAS A PAS

La collection « **GUIDES PRATIQUES** » a pour vocation de recueillir des ouvrages consacrés à un thème précis et limité.

Huit titres sont actuellement disponibles dans cette collection ; certains disposent d'une disquette sur Apple II (*).

(*) Méthodes de calcul numérique — tomes 1 et 2 — Claude Nowakowski
(*) Visicalc sur Apple — Hervé Thiriez
Visicalc sur TRS-80 — Hervé Thiriez
— APL sur TRS-80 — Claude Nowakowski
— CP/M pas à pas — Alain Pinaud
— LISP sur Apple II — Nicole Bréaud Pouliquen
— La réalisation des programmes — Michel Benelfoul

Autres ouvrages relatifs au système CP/M :

— CP/M mot par mot — Yvon Dargery
Collection « MEMENTOS »

— Le système CP/M pour Z-80 — Fabienne et Philippe Gysel.
Editests, diffusé par PSI.

RAPPELS

Les séries :

En fait, il faudrait parler de niveaux, puisque la couleur attachée à chaque ouvrage permet de situer la « force » de celui-ci selon le code suivant :

Série VERTE : ouvrage d'initiation ne nécessitant que des connaissances de base.
Série BLEUE : suppose une connaissance élémentaire du sujet traité.
Série ROUGE : ouvrage d'approfondissement, niveau de complexité moyen.
Série NOIRE : ouvrage d'approfondissement, niveau de complexité élevé.

Les collections :

Les ouvrages d'Edition du PSI, actuellement au nombre de 100, sont répartis en collections :

« **LANGAGES** », « **MATERIELS** », « **PROGRAMMES** », « **GUIDES PRATIQUES** », « **MEMENTOS** », « **UTILISATIONS DE L'ORDINATEUR** » et pour l'initiation, outre quelques livres hors collection, « **... POUR TOUS** ».

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective », et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article 40).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

© *Editions du P.S.I. Parc Industriel Nord - Bâtiment 9 - 77200 Torcy (France)*

ISBN : 2-86595-051-4

1983

GUIDE PRATIQUE

CP/M PAS A PAS

ALAIN PINAUD

EDITIONS DU P.S.I.
1983

P R E S E N T A T I O N

Alain PINAUD, *37 ans, est ingénieur informaticien chez le constructeur Français d'ordinateurs CII-HOMEYWELL-BULL, où il participe à l'étude et au développement de systèmes spéciaux, tant sur le plan matériel que logiciel.*

Conseiller technique pour la revue L'Ordinateur Individuel. il est d'autre part, rédacteur en chef du magazine TRACE. dédié au TRS-80.

Le sigle ***CP/M*** *tel qu'il est utilisé dans ce livre est une marque déposée de Digital Research, Inc, de Pacific Grove, Californie (USA).*

S O M M A I R E

Pages

INTRODUCTION 7

I - A VOS MARQUES...

Qu'est-ce qu'un système d'exploitation ? 9
Qu'est-ce que CP/M ? 9
Fonctions remplies par un système d'exploitation 10

Petit historique de CP/M 11

Structure du système CP/M 12
Le chargement 12
Organisation mémoire 13
Commandes offertes par CP/M
Commandes résidentes 15
Commandes transitoires 15

Conclusions sur CP/M 15

II - PRETS ?

Configuration minimale 17
Fichiers et organes périphériques 17
Syntaxes des commandes 18
Codes de contrôle 19

III - PARTEZ !

DIR 21
USER 23
STAT 24
REN 32
TYPE 33
PIP (premier niveau) 34
d: 37
ERA 38
DUMP 39
ED (premier niveau) 40
SUBMIT 45
ASM 47
LOAD 52
DDT (premier niveau) 53
SAVE 58
XSUB 61
MOVCPM 63
SYSGEN 65

Pages

IV - ... ENCORE UN PETIT EFFORT !

ED (second niveau) 69
PIP (second niveau) 81
DDT (second niveau) 90

ETES-VOUS BIEN ARRIVES ?

ANNEXES

- 1 Affectation mémoire 100
- 2 Description octet entrée/sortie 101
- 3 Points d'entrées BDOS 102
- 4 Points d'entrées BIOS 104
- 5 Description du FCB 105
- 6 Occupation disquette 106
- 7 Format HEX d'Intel 107
- 8 l'Assembleur 109
- 9 Structure des fichiers CP/M 114
- 10 BASIC sous CP/M 115
- 11 CP/M-80 et CP/M-86 117
- 12 Code ASCII 119
- 13 Bibliographie 120
- 14 Résumé commandes CP/M 121

INTRODUCTION

Il existe quantité d'ouvrages sur le système d'exploitation CP/M. Celui-ci en est un autre, rédigé dans un autre esprit. La plupart des ces ouvrages sont des ouvrages de référence : ils contiennent tout, mais ils nécessitent souvent un savoir préalable, et un lecteur novice ne saura pas toujours par quel bout les aborder... Dans ceux-ci, les commandes CP/M sont hiérarchisées,triées, classées alphabétiquement... Malheureusement, les cellules du cerveau humain ne semblent pas être classées alphabétiquement, et cette présentation, bien que d'une consultation aisée, ne facilite pas toujours la compréhension du lecteur.

Vu sous cet angle, il semblera régner la plus profonde anarchie dans notre ouvrage ! Celui-ci, se voulant essentiellement pratique, aborde les commandes CP/M par niveau de compréhension. Pour donner un simple exemple, il sera nécessaire, avant de pouvoir assembler un programme, de le construire par l'éditeur.

Nous sommes donc partis du principe fondamental, que le lecteur était installé devant un ordinateur fonctionnant sous CP/M, et qu'il suivait la progression donnée notamment dans les chapitres 3 et 4, en pratiquant les exemples qui y sont décrits.

Afin de minimiser le nombre d'erreurs, toujours possibles,ces exemples ont été reproduits tels qu'ils avaient été imprimés par l'ordinateur ayant servi à nos essais.

Les notions d'exploitation de l'assembleur CP/M décrites dans cet ouvrage, demandent une certaine initiation à ce langage, de la part du lecteur. Toutefois, ce dernier pourra toujours suivre les exemples décrits - même s'il n'en comprend pas toujours la profonde signification - et consulter la liste des ouvrages de cette collection s'il est désireux d'en savoir plus sur l'assembleur.

A VOS MARQUES...

QU'EST-CE QU'UN SYSTEME D'EXPLOITATION DE DISQUETTES ?

Le système d'exploitation (Operating System, ou OS en anglais) est un programme - la plupart du temps écrit en langage machine pour des raisons de performances - servant d'interface logicielle entre l'ordinateur et son utilisateur.

Tout dialogue entre l'homme et la machine passe - directement ou non - par le canal du système d'exploitation. Lorsque l'ordinateur est équipé de mémoires de masse tels que disques magnétiques ou disquettes (ou même minidisquettes), le système d'exploitation prend alors le nom de DOS (Disk Operating System) ou SED (Système d'exploitation de Disquettes en français).

Tout ordinateur tant soit peu évolué, se doit d'être équipé d'un SED dès l'instant où l'utilisateur désire exploiter sérieusement des fichiers.

Qu'est-ce que CP/M ?

CP/M est le nom du SED le plus répandu dans la catégorie des micro-ordinateurs construits autour du microprocesseur 8080 ou équivalent (8085 ou Z80). Ce système d'exploitation est connu pour sa simplicité d'emploi, sa fiabilité et son adaptabilité aux différents ordinateurs utilisant ce microprocesseur.

Nécessité d'un système d'exploitation

Devant l'importance grandissante du stockage offert par les disques ou les disquettes (de 80.000 à 10 millions de caractères selon le type employé), l'utilisateur se rend très vite compte de l'énormité du travail qui lui serait imputé s'il devait s'occuper lui-même de la gestion de l'espace disque et du dialogue avec l'environnement extérieur (organes périphériques).

C'est pourquoi l'un des buts du système d'exploitation est de simplifier au maximum la tâche de l'utilisateur, en prenant en charge la résolution de tous les problèmes ingrats et routiniers que celui-ci ne tient pas particulièrement à connaître... (heureusement, nous n'avons pas à demander l'avis des ordinateurs !).

Fonctions remplies par un système d'exploitation

Aussi, pouvons-nous nous attendre à ce que tout système d'exploitation digne de ce nom, offre les services suivants :

- un accès facile et rapide aux fichiers de données ou de programmes.

En pratique, il suffira de spécifier au SED le nom d'un fichier pour que celui-ci soit capable de l'extraire rapidement et sans risques d'erreurs, parmi les millions de bits enregistrés sur le support magnétique.

Peu importe à l'utilisateur (en général !) de connaître l'endroit exact où est physiquement implanté le fichier (c'est le SED qui se charge de cela), le principal est d'y avoir accès en cas de besoin. C'est un peu comme lorsque vous allez dans une librairie demander un ouvrage bien précis : si vous deviez le chercher vous-même, cela prendrait sûrement un certain temps... Le libraire, par contre, vous le fournira dans les secondes qui suivent.

A l'inverse, le système d'exploitation devra être capable de stocker (là où il y a de la place...) le fichier que l'utilisateur lui confie, même s'il doit le morceler pour occuper les quelques "trous" qui existent sur la disquette.

La gestion de l'espace disque est une caractéristique très importante des SED.

- le traitement des opérations classiques de maintenance des fichiers.

Il arrive fréquemment que l'utilisateur éprouve le besoin de connaître le nom des fichiers disponibles sur sa disquette. Pour cela, il aimerait en consulter le CATALOGUE (directory).

Tel fichier n'est plus nécessaire : supprimons-le. Tel autre porte mal son nom : nommons-le autrement. Ce fichier très important risque d'être détruit par mégarde : protégeons-le ou recopions-le sur une disquette de sauvegarde. Reste t-il encore de la place sur la disquette pour y placer ce petit programme ?

A toutes ces tâches, le système d'exploitation est capable de répondre, moyennant une simple commande.

- un dialogue facile avec les organes périphériques (entrées/sorties).

Etant données les différences de comportement existant d'un organe périphérique à un autre, il serait pénible, voire même rebutant pour l'utilisateur, de devoir s'adapter à chacune d'elles.

Pour résoudre efficacement ces problèmes, le système d'exploitation contient des programmes spécifiques aux organes périphériques appelés "drivers" ou "conducteurs", qui se chargent d'aplanir (d'intégrer) toutes ces différences de comportement, en offrant à l'utilisateur des commandes évoluées permettant une banalisation des opérations d'entrées/sorties. Aux yeux de ce dernier, le dialogue avec les organes périphériques se trouve "standardisé", ce qui signifie que par une simple commande, tel message apparaissant habituellement sur l'écran vidéo pourra être envoyé, par exemple, vers l'imprimante ou, pourquoi pas, sur le disque sous la forme d'un fichier. Pourtant, ces organes périphériques sont de natures totalement différentes.

La seule précaution à prendre concernera le sens du transfert propre à chaque organe (pas question d'imprimer un message sur le clavier ou d'entrer une commande par l'imprimante... !).

Enfin, **certains SED** tel que **MP/M** par exemple, ainsi que les systèmes d'exploitations des gros ordinateurs, sont capables d'**exécuter plusieurs tâches en simultanéité apparente** (contexte multi-tâches ou multi-traitements). En réalité, l'ordinateur - à moins qu'il soit équipé de plusieurs unités centrales (multiprocesseurs) - exécute séquentiellement des fragments plus ou moins entrelacés de ces différentes tâches.

Ce n'est pas le cas du CP/M et nous n'aborderons pas ce domaine qu'il fallait pourtant citer au nombre des principales caractéristiques des systèmes d'exploitation.

PETIT HISTORIQUE DE CP/M

Le système d'exploitation CP/M (Control Program for Microcomputers) est, comme son nom l'indique, un programme de contrôle pour micro-ordinateurs.

Il a vu le jour sous une forme d'abord simplifiée, aux alentours de l'année 1974, date à laquelle apparut la première unité de disque souple (floppy-disk). Quand au premier microprocesseur 8 bits évolué, le 8080 d'Intel, il était né un an auparavant.

A cette époque, qui n'est pourtant pas très lointaine, la machine de dialogue en vigueur sur l'ordinateur était la Télétype modèle 33 (probablement née des siècles avant...), réputée pour sa lenteur et son bruit assourdissant. Bon nombre d'informaticiens qui ont vécu cette époque, sont encore pris de tremblements spasmodiques à la simple évocation de ce souvenir !! Fichiers et programmes quant à eux, se rencontraient sous la forme de rubans perforés.

Gary Kildall, qui travaillait alors en tant que consultant chez Intel, eût la chance de se trouver à la croisée du microprocesseur 8080 (équipant l'ordinateur MDS 800 d'Intel) et des premières unités de disques souples. Il conçut alors, à des fins personnelles, une ébauche de système d'exploitation adapté à ce tout nouvel organe périphérique, qu'il appela CP/M.

En 1975, Intel ne semblant pas s'intéresser outre mesure à ce type de logiciel, Gary Kildall obtint les premières licences commerciales de CP/M avec trois entreprises américaines qui produisaient alors les tous premiers micro-ordinateurs à base de disques souples.

De plus en plus de constructeurs se montrant intéressés par CP/M, des problèmes d'adaptation à ces différents ordinateurs ne tardèrent pas à se poser.

C'est à ce moment qu'apparut la notion de **BIOS**, cette *portion de CP/M dépendant du matériel sur lequel il est implanté*. En dehors de cette partie spécifique pouvant être facilement configurée et adaptée au gré des utilisateurs, le reste du système CP/M est indépendant du matériel, pourvu que ce dernier soit piloté par un microprocesseur 8080 (ou compatible).

Pendant cette période, le CP/M se "muscla" un peu et les programmes utilitaires - courants maintenant - tels que l'éditeur (**ED**), l'assembleur (**ASM**) et le système de mise au point (**DDT**) furent inclus sur la disquette système pour devenir partie intégrante de CP/M.

En 1976, un accord fut passé avec **Imsai Inc**, qui amorça la diffusion des premiers **ordinateurs individuels.**

La même année, Gary Kildall fonda **Digital Research**, société chargée de diffuser et de supporter CP/M.

1973 - *microprocesseur 8080*
1974 - *première unité de disque souple*
1975 - *premières licences commerciales de CP/M*
1976 - *BIOS - Imsai - Digital Research CP/M version 1.3*
1981 - *CP/M version 2.2*

Depuis ce *mariage entre un microprocesseur et une disquette*, CP/M ne cesse d'évoluer et **la version actuelle** (2.2) **semble s'orienter vers une gestion plus efficace des grands espaces apportés par les disques durs.**

STRUCTURE DU SYSTEME CP/M

Le chargement

Il est demandé peu de choses aux ordinateurs devant recevoir CP/M : ils doivent être équipés d'une mémoire morte (ROM ou PROM ou EPROM) contenant un petit programme de chargement d'un secteur de la disquette ; le reste est *constitué de mémoire vive (RAM).*

Toutefois, et afin d'être entièrement compatible avec une version standard de CP/M, la mémoire vive doit commencer à l'adresse 0.

Mais à la mise sous tension d'un système à base de microprocesseur 8080 ou équivalent, le contrôle est donné à l'instruction située à l'adresse 0 de la mémoire qui doit donc être en mémoire morte... Il semble donc y avoir un conflit matériel.

Pour répondre à ces exigences, nous trouvons deux procédés.

Le premier utilise une mémoire morte située à l'adresse 0. Lorsque sa fonction d'initialisation est remplie, elle s'escamote au moyen d'un dispositif matériel, et est remplacée par de la mémoire vive.

Le second procédé est également basé sur un dispositif matériel qui, à la mise sous tension, force le microprocesseur à exécuter une instruction se trouvant à une adresse différente de zéro. Cela peut être l'adresse d'une ROM située en fond de mémoire. A l'adresse zéro, se trouve donc la mémoire vive.

Bref, le contrôle est donné au programme se trouvant en mémoire morte, et celui-ci n'a qu'un objectif : *charger en mémoire vive le contenu du premier secteur se trouvant sur la disquette CP/M* (piste 0, secteur 1). Cette opération se nomme le "**bootstrap**" de la disquette ou "**démarrage à froid**" (cold boot).

Un branchement est ensuite exécuté vers le petit programme venant d'être lu en mémoire, appelé "chargeur".

Les instructions de ce programme ont une fonction bien précise : charger en mémoire vive (en fond de mémoire généralement), le contenu des deux premières pistes de la disquette CP/M (moins le secteur 1). Ces pistes contiennent le système CP/M proprement dit (**figure 1**).

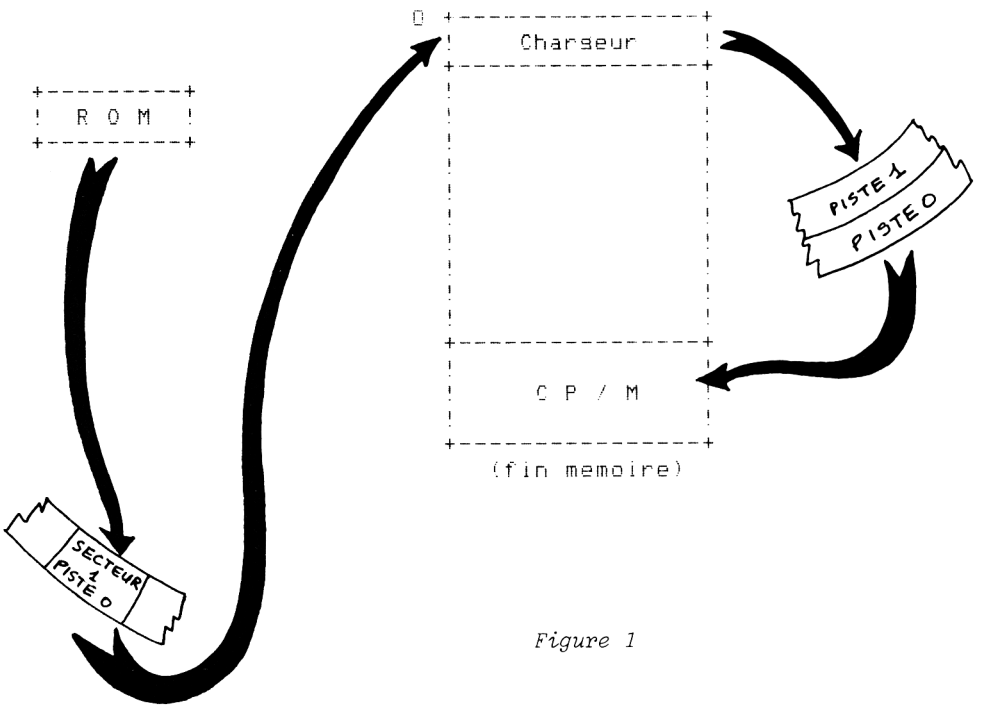


Figure 1

Organisation mémoire

La zone correspondant aux 256 premiers octets de la mémoire (adresse 0 à 0FFH) est appelée **page 0.**

C'est la zone de référence du système, contenant un certain nombre de données et de pointeurs situés à des adresses fixes indépendantes du système ou de la configuration.

La zone située après la page 0 (commençant en 100H) est appelée **zone TPA** (Transient Program Area = zone des programmes transitoires). *C'est la mémoire utilisateur où seront chargés les programmes et les commandes transitoires de CP/M.* Selon la configuration du système, cette zone sera plus ou moins grande.

A la suite, se trouve **CCP** (Console Command Processor = analyseur des commandes console). *Cette portion de CP/M est chargée d'analyser et d'exécuter les commandes issues de la console* (vos commandes). Elle contient les commandes résidentes CP/M et, le cas échéant, charge en zone TPA les commandes transitoires.

Lorsque la zone TPA ne suffit pas à contenir un programme, CCP peut être détruite pour lui laisser la place. A la fin

d'exécution du programme, il sera donc nécessaire d'exécuter ce que l'on nomme un "**démarrage à chaud**" (warm boot) *ayant pour but de recharger en mémoire la zone CCP.*

Après CCP, se trouve le **FDOS** (Functionnal Disk Operating System = SED fonctionnel) *qui est le noyau des routines constituant CP/M.* Cette zone est découpée en deux parties : le **BDOS** qui suit la zone CCP et le **BIOS** qui termine la zone FDOS.

La **zone BDOS** (Basic Disk Operating System = SED de base) *contient le système de gestion des fichiers disquette.* C'est un ensemble de routines propres aux opérations de lecture/écriture des fichiers.

La **zone BIOS** (Basic Input Output System = système d'entrée/sortie de base) *est la seule portion de CP/M devant être adaptée à l'ordinateur sur lequel il est implanté.* Cette portion de programme contient en effet tous les "pilotes" ou "conducteurs" (drivers) des organes périphériques reliés à l'ordinateur qui, comme chacun le sait, sont gérés différemment d'un système à l'autre. En partant du CP/M standard délivré par Digital Research, il incombera donc au constructeur (ou à la société qui propose CP/M sur un système donné) de réaliser l'écriture du BIOS et de l'intégrer sur la disquette CP/M commercialisée avec la machine (**figure 2**).

```
0000H +---------------------------+
      !  Page 0 reservee a CP/M   !
      !        (256 octets)       !
0100H +---------------------------+
      !                           !   Transient
      /          T P A            /   Program
      /   (depend de la confis)   /   Area
      +---------------------------+
      !                           !   Console
      !          C C P            !   Command
      !        (2Ko env)          !   Processor
      +---------------------------+
      !                           !   Basic
      !          B D O S          !   Disk
      !        (3Ko env)          !   Operating
F     !                           !   System
D     +---------------------------+
O     !                           !   Basic
S     !          B I O S          !   Input/output
      !        (0,5 a 1Ko)        !   Operating
      !                           !   System
      +---------------------------+
      !///////////////////////////!
      !      Zone optionnelle     !
      +---------------------------+
            (fin de memoire)
```

Figure 2

Commandes disponibles sur CP/M

Comme nous venons de l'entrevoir dans les lignes précédentes, CP/M offre deux types de commandes à l'utilisateur : les **commandes résidentes** (*toujours présentes en mémoire*) et les **commandes transitoires** (prenant souvent la forme de programmes

"utilitaires") *présentes sur disquette et appelées en zone TPA pour exécution.*

Cette philosophie n'a qu'un but : réduire au maximum l'occupation mémoire du système CP/M en déportant sur disque les commandes n'étant pas de première nécessité.

Voici la liste de ces commandes CP/M :

Commandes résidentes.

DIR	*consultation du catalogue de la disquette*
ERA	*destruction de fichiers*
REN	*changement des noms de fichiers*
SAVE	*sauvegarde mémoire -> disque*
TYPE	*affichage du contenu de fichiers ASCII*
USER	*partitionnement de la disquette*
d:	*changement disquette implicite*

Commandes transitoires.

ASM	*appel de l'assembleur*
DDT	*appel du programme de mise au point*
DUMP	*affichage hexa d'un fichier binaire sur disque*
ED	*appel de l'éditeur*
LOAD	*conversion de programmes en commandes*
MOVCPM	*configurateur mémoire*
PIP	*échange entre fichiers et périphériques*
SUBMIT	*enchaînement automatique (batch)*
STAT	*états des fichiers et des disquettes*
	configurateur périphériques
SYSGEN	*génération de système*
XSUB	*extension SUBMIT*

Conclusions sur CP/M

CP/M est un système d'exploitation **simple** d'emploi.

Loin des SED ultra-sophistiqués offrant des possibilités la plupart du temps jamais exploitées par l'utilisateur étant donnée leur complexité, CP/M se situe comme un système simple, offrant un jeu de commandes suffisant pour couvrir un vaste éventail d'applications.

CP/M est un système **fiable.**

Cette qualité, qui est intrinsèquement liée à la complexité du système est primordiale tout en étant rassurante pour l'utilisateur.

Bien qu'un peu désuet sur certains points, CP/M possède cette maturité l'immunisant contre tous ces petits problèmes de jeunesse, fréquents sur d'autres systèmes d'exploitation.

CP/M est **très répandu.**

Etant donné son âge, il est normal qu'on le trouve implanté sur une grande quantité d'ordinateurs. Son **faible prix** y est également pour quelque chose. Cela favorise une autre qualité, celle de rendre les programmes écrits sous CP/M transportables d'une machine à l'autre, ce qui n'est pas à négliger, autant par les utilisateurs que par les concepteurs de logiciels.

Sans cette grande diffusion, la transportabilité n'aurait en effet aucun sens. Il en résulte pour l'utilisateur, un **accès à une énorme bibliothèque logicielle** d'une très grande diversité.

CP/M est **facilement adaptable aux ordinateurs** équipés d'un microprocesseur 8080 ou compatible.

Ce point présente un double avantage : l'utilisateur n'aura aucune difficulté à apprendre et à maîtriser ce système d'exploitation universellement connu, qu'il retrouvera aussi sur d'autres machines ; le constructeur pourra offrir rapidement et pour un bas prix un système dont l'étude se bornera à la seule écriture du BIOS et de l'utilitaire de formattage.

PRETS ?

CONFIGURATION MINIMALE

Pour pouvoir pratiquer les exemples décrits dans cet ouvrage, il est nécessaire de posséder un système CP/M implanté sur un ordinateur dont la configuration minimale est la suivante :

- 16 ou 20K octets de mémoire vive (selon version 1 ou 2 de CP/M)
- un organe d'entrée (clavier)
- un organe d'affichage (écran vidéo)
- une disquette ou mini-disquette

Optionnellement, il sera souhaitable de disposer d'une capacité mémoire de 48 ou 64K octets, d'une seconde unité de disquette ou minidisquette, ainsi que d'une imprimante.

FICHIERS ET ORGANES PERIPHERIQUES

Un fichier est une collection d'articles. Ce peut être une liste téléphonique, un ensemble de recettes de cuisine ou un programme exécutable.

En CP/M, un fichier est **désigné par un nom** composé de 1 à 8 caractères alphanumériques ou de quelques caractères spéciaux. Afin de faciliter leur exploitation, il est courant de classer les fichiers par **types** : les fichiers en langage BASIC, les fichiers en langage source assembleur, les fichiers contenant une liste de programmes, etc...

Le nom du fichier pourra donc optionnellement être suivi d'un nom de type sur 1 à 3 caractères, un point "." séparant les deux.

Bien que le nom du type puisse être choisi par l'utilisateur, quelques-uns d'entre eux sont implicitement connus de certaines commandes CP/M. Voici la liste des principaux :

.COM *fichier de commande, directement utilisable sous CP/M. Créé par la commande LOAD à partir d'un fichier de type .HEX*

.HEX *fichier objet créé par l'assembleur (format hexadécimal Intel). Peut être chargé et exécuté par DDT (metteur au point)*

.PRN *fichier liste créé par l'assembleur*

.ASM *fichier contenant le source assembleur d'un programme. Créé par l'éditeur ED*

.BAK *fichier de sauvegarde créé par l'éditeur ED*

.SUB *fichier de commandes (batch) créé par l'éditeur et exploité par la commande SUBMIT*

.$$$ *fichier temporaire créé par certains processeurs (ED par exemple)*

La disquette - ou minidisquette - bien qu'organe périphérique à part entière - n'est pas considérée ainsi par CP/M qui la voit comme faisant partie intégrante du système, au même titre que la mémoire.

Celui-ci par contre, distingue 4 organes périphériques auxquels sont attribués des noms immuables :

CON : *console écran/clavier*
RDR : *organe d'entrée de données (lecteur rubans)*
PUN : *organe de sortie de données (perfo rubans)*
LST : *organe de sortie de listes*

Ces organes reçoivent l'appellation d'organes "logiques" et correspondent à des fonctions et non à des périphériques précis.

Toutefois, ces derniers ont également un nom :

CRT : *écran cathodique*
TTY : *télétype*
LPT : *imprimante*
etc...

et reçoivent l'appellation d'organes "physiques". Une instruction CP/M (**STAT**) permet d'affecter un organe physique à un organe logique et un programme utilitaire (**PIP**) peut effectuer des échanges entre fichiers disque et organes logiques. A chacun de ces organes physiques, devra correspondre un programme pilote (driver) dans le BIOS.

SYNTAXE DES COMMANDES

Les commandes CP/M obéissent à une syntaxe stricte mais relativement simple. Elles peuvent être introduites en caractères majuscules ou minuscules.

Les noms de fichiers (ou de commandes) sont formés de 1 à 8 caractères alphanumériques, et ne doivent pas contenir d'espace ainsi que les symboles suivants :

. , : ; = ? * [ ]

(nous verrons toutefois une exception un peu plus loin, en ce qui concerne les symboles "*" et "?").

Exemples : PROG , A2Pa , PROGRAMM , fichel.tel

Un **fichier** est complètement défini sous cette forme :

<nom disquette><nom fichier>.<type>

Les signes "<" et ">" sont utilisés comme délimiteurs syntaxiques et ne sont pas inclus dans la définition.

Le nom de la disquette est représenté par une lettre de A à P, suivie immédiatement du signe ":". La première disquette se nommera A:, la seconde B: etc...

Le système considère toujours la disquette A: comme étant implicite, aussi, il n'est pas nécessaire de préciser son nom dans une référence de fichier. Par contre, tout fichier appelé d'une autre disquette devra avoir son nom précédé du nom de cette disquette.

Une commande CP/M permet toutefois de modifier l'affectation de la disquette implicite.

Exemple : B:PROG, b:toto.asm

Le type est formé de 1 à trois caractères alphanumériques et obéit aux mêmes restrictions que le nom du fichier.

CP/M permet de manipuler non seulement des fichiers mais aussi des groupes de fichiers, comme par exemple, tous les fichiers dont la seconde lettre est un C et dont le type est ".ASM"... C'est la notion de référence ambigüe.

Lorsqu'un fichier est clairement défini (nom et type formés de symboles alphanumériques à l'exclusion des quelques restrictions évoquées plus haut), nous dirons qu'il possède une **référence explicite.**

Lorsqu'un fichier comporte dans son nom ou dans son type, les symboles "*" ou "?", nous dirons qu'il possède une **référence ambigüe.**

Lorsqu'il est employé dans le nom ou dans le type d'un fichier, le symbole "?" signifie "peut être remplacé par n'importe quel caractère". Lorsque le symbole "*" figure à la place du nom et/ou du type d'un fichier, cela signifie "peut être remplacé par n'importe quel nom et/ou n'importe quel type".

Exemple 1 : ??D?.ASM peut désigner les fichiers ABDE.ASM, DEDE.ASM, CADO.ASM, etc...

Exemple 2 : ??CD.* peut désigner les fichiers ABCD.ASM, CDCD.HEX, P1CD, etc...

Exemple 3 : *.* désigne tous les fichiers de la disquette.

Dans les exemples des chapitres suivants, nous désignerons ces deux formes par : <ref explicite> et <ref ambigüe>.

CODES DE CONTROLE

CP/M ainsi que les programmes associés, tel que l'éditeur par exemple, reconnaissent un certain nombre de caractères de contrôle, ainsi appelés parce qu'ils sont générés à l'aide de la touche "control" (CTRL) du clavier de la console.

Sur l'écran de la console ou dans les listes des programmes, le caractère "control" est souvent matérialisé par le graphisme "flèche en haut". Ainsi, ↑C désignera un appui simultané sur les touches CTRL et C du clavier.

Parfois, certains caractères de contrôle ont une action directe sur CP/M et ne sont pas affichés.

Voici la liste de ces codes ainsi que les fonctions correspondantes :

CTRL et	Code hexa	Action
C	03	Démarrage à chaud du CP/M
E	05	Retour ligne "local" sans fin de commande
H	08	Annulation caractère (BS)
I	09	Tabulation (8 colonnes)
J	0A	Nouvelle ligne (LF)
L	0C	Représente les codes RC et LF. Utilisé avec ED
M	0D	Retour chariot (CR)
P	10	Bascule de recopie d'écran sur imprimante (on/off)
R	12	Re-affiche la ligne
S	13	Arrêt temporaire affichage
U	15	Annulation de ligne
X	18	Annulation de ligne
Z	1A	Indique la fin d'une chaîne. Utilisé avec ED et PIP

PARTEZ !

Ce chapitre est essentiellement pratique. La plupart des exemples donnés ci-dessous, ont été réalisés sur un système équipé de la version 2.2 de CP/M.

La progression des éléments de ce chapitre a été étudiée de façon à ce que vous vous posiez le minimum de questions...

Il est donc conseillé de mettre votre système sous tension, de placer cet ouvrage devant votre clavier, et d'essayer directement sur VOTRE ordinateur, toutes les manipulations qui vont être décrites par la suite.

A l'initialisation du système, CP/M affiche sur l'écran :

```
20k CP/M vers 2.2

A>
```

La valeur "20K" dépend de la configuration employée et il se peut que la vôtre soit différente. En principe, la version originale du CP/M est configurée au minimum de mémoire. Charge à l'utilisateur de faire le nécessaire (grâce à la commande MOVCPM) s'il désire utiliser une autre configuration mieux adaptée à son système.

Le message affiché se termine par "**A>**". Cela signifie que la disquette (ou minidisquette) référencée "A" est affectée au CP/M.

Toutes les opérations relatives aux disquettes ne mentionnant pas une référence explicite (A: , B: , etc) seront effectuées sur A:.

DIR - commande de listage du catalogue

Tous les systèmes d'exploitation possèdent une *commande permettant de consulter le CATALOGUE d'une disquette* (DIRECTORY en anglais). Ce catalogue contient la liste de tous les programmes disponibles sur ce support magnétique. La commande CP/M correspondante est **DIR.** Frappons-la, en la terminant par un appui sur la touche CR (ou RETURN).

```
A>DIR
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : DUMP     PRN
A>
```

La liste des programmes normalement fournis avec le CP/M standard est affichée à raison de 4 programmes par ligne, chacun d'eux étant séparés par le symbole ":". Sur la liste ci-dessus, la première ligne nomme les 4 programmes :

MOVCPM.COM
PIP.COM
SUBMIT.COM
XSUB.COM

Notez que le "." séparant le nom du fichier de son type n'apparaît pas dans la liste. La version 1.4 du CP/M offrait une présentation moins pratique (un nom par ligne). Un examen détaillé montre que DIR n'apparaît pas dans la liste... Il s'agit donc d'une **commande résidente** de CP/M.

Dans l'hypothèse où vous possédez une seconde unité de disquette (appelée B:), essayez :

```
A>DIR B:
NO FILE
A>
```

Notre disquette étant vierge à l'origine, l'affichage ci-dessus montre qu'aucun fichier n'y est présent (NO FILE).

Grâce à cette simple commande, il est possible de contrôler la présence d'un fichier particulier sur le disque :

```
A>DIR MOVCPM.COM
A: MOVCPM   COM
A>
```

... ou son absence :

```
A>DIR TOTO
NO FILE
A>
```

On peut également *extraire de la liste un groupe de fichiers* répondant à certains critères syntaxiques (référence ambigüe), comme par exemple, les fichiers commençant par la lettre D, possédant un maximum de quatre caractères et ayant le type ".COM" :

```
A>DIR D???.COM
A: DDT      COM : DUMP     COM
A>
```

Dans notre cas, deux fichiers (DDT.COM et DUMP.COM) répondent à ces critères.

Un autre exemple basé sur le même principe, mais dont le type peut être quelconque :

```
A>DIR D???.*
A: DDT      COM : DUMP      COM : DUMP      ASM : DUMP      PRN
A>
```

fait apparaître deux fichiers supplémentaires (DUMP.ASM et DUMP.PRN).

EN RESUME

DIR -commande de listage du catalogue	
SYNTAXE	**EXEMPLES**
DIR <>	DIR
<d:>	DIR B:
<ref explicite>	DIR MOVECPM.COM
<ref ambigue>	DIR B:D???.*
commande residente	

USER - commande de changement de zone utilisateur

La version 2 du CP/M est particulièrement orientée vers la gestion des grands espaces disque. Il peut donc paraître intéressant à différents utilisateurs de s'*affecter des zones indépendantes sur le même support* magnétique, d'où la notion de "zones utilisateurs" (USER AREA).

Monsieur Dupont, par exemple, ne souhaite voir que ses propres programmes lorsqu'il exécute la commande DIR sur le disque commun à tous les membres de sa société. Il ne tient nullement à y voir ceux de son "brouillon" de collègue, monsieur Durand ! Par contre, il aimerait avoir un accès temporaire aux astucieux programmes de monsieur Malin.

En version 2.2, il est possible de déclarer 16 zones différentes, numérotées 0 à 15. La zone 0 est automatiquement affectée à la mise sous tension. C'est celle contenant les programmes que nous avons examinés plus haut. Les autres zones sont vides au départ.

Pour "passer" dans la zone 1, par exemple, il suffit de frapper :

```
A>USER 1
A>
```

La commande suivante :

```
A>DIR
NO FILE
A>
```

ne nous incite pas à rester dans cette zone pour le moins désertique... Mieux vaut retourner dans des terres plus hospitalières :

```
A>USER 0
A>DIR *.PRN
A: DUMP      PRN
A>
```

Nous verrons un peu plus tard - lorsque nous serons mieux affirmés - comment exploiter ces zones utilisateurs. Contentons-nous pour l'instant de cette brève incursion.

EN RESUME

USER - commande de chansement de zone utilisateur	
SYNTAXE	**EXEMPLES**
USER <0 a 15>	USER 0 USER 15
commande residente - version 2 uniquement	
voir complement ---> SAVE	

STAT - commande de listage des informations sur les disques et les programmes
- modification des affectations périphériques

La commande suivante va nous permettre d'obtenir davantage d'*informations STATistiques* sur les disquettes, les fichiers qui y sont contenus et les organes périphériques connectés au système.

```
A>STAT
A: R/W, Space: 156k
B: R/W, Space: 241k


A>
```

Lorsque la commande **STAT** est employée sans paramètres, l'état de toutes les disquettes actives explicitement déclarées est donné : R/W signifie que la disquette peut être lue et écrite (Read/Write) ; nous voyons d'autre part que l'espace (space) disponible sur la disquette A est de 156K octets (disquette système), contre 241K octets pour la disquette B (non système).

La disquette B est reconnue active car nous y avons effectué au moins une opération depuis la mise en route du système (DIR B:). Sans cela, seul l'état de la disquette A nous aurait été donné.

STAT suivi d'un nom de disquette ne rend que l'espace disponible sur cette disquette :

```
A>STAT A:

Bytes Remaining On A: 156k

A>STAT B:

Bytes Remaining On B: 241k

A>
```

Il peut être utile de connaître la configuration des organes périphériques affectés à CP/M. Le mot-clé **"DEV :"** (DEVices ou périphériques) servira dans ce cas de paramètre à la commande **STAT :**

```
A>STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is TTY:

A>
```

CP/M considère les organes périphériques (autres que les disquettes) selon deux niveaux : le niveau logique et le niveau physique.

Lorsque le système génère un listing, il affecte cette opération à l'organe logique **LST:** (LISTing). Mais physiquement, cet organe peut être une imprimante, un écran ou même un perforateur de ruban.

La commande ci-dessus nous renseigne sur les affectations actuelles :

- l'organe physique **CRT:** (clavier/écran vidéo) est affecté à l'organe logique **CON:** (console) ;

- l'organe physique **TTY:** (Télétype) est affecté aux organes logiques de sortie de listes (**LST:**), de lecture (**RDR:**) et de perforation (**PUN:**) de rubans de papier.

En fait, l'usage de ces mots-clés remonte à l'époque du développement initial de CP/M, époque glorieuse où la télétype était reine.

Cet organe périphérique comprenait - et comprend toujours - un clavier, un système d'impression de papier et un lecteur/perforateur de ruban. Le mot-clé CRT: est apparu plus récemment, remplaçant ainsi la télétype qui faisait également office de console !

Les différentes affectations organes physiques/logiques seront examinées plus loin (**STAT VAL:**).

Mais dans l'immédiat, nous désirons en savoir davantage sur les disquettes. Frappons la commande suivante (**STAT A:DSK:** si vous ne possédez qu'une seule disquette) :

```
A>STAT B:DSK:

    B: Drive Characteristics
 1944: 128 Byte Record Capacity
  243: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block
   26: Sectors/ Track
    2: Reserved Tracks

A>
```

et les caractéristiques de la disquette B: sont décrites. Pour obtenir une liste des caractéristiques de toutes les unités actives, il suffira de frapper : **STAT DSK:** sans mentionner de nom de disquette.

Ces informations, bien que relativement techniques, ne doivent pas émouvoir les praticiens que nous sommes !

Vous souvenez-vous des zones utilisateurs ? Il est possible de connaître celles qui sont définies ainsi que celle qui est active à l'instant présent :

```
A>STAT USR:

Active User : 0
Active Files: 0
A>
```

Pour l'instant, seule la zone 0 est active et définie. Mais ce n'est pas notre dernier mot !

La commande **STAT** nous fournit aussi quelques informations pratiques lorsque nous avons un "trou" de mémoire (à condition toutefois de se souvenir dú mot-clé **VAL:** qui est d'une grande VALeur...) :

```
A>STAT VAL:

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>
```

Les deux premières lignes affichées par la commande **STAT VAL:** concernent des paramètres que nous allons examiner par la suite. Quant aux deux lignes suivantes, nous les connaissons déjà.

Nous trouvons dans les quatre dernières lignes, les différentes affectations possibles des organes périphériques évoquées plus haut. Nous voyons par exemple, que la console peut être :

TTY:, **CRT:**, **BAT:** ou **UC1:**. Si cela n'évoque rien en vous, voici la signification des différents mots-clés des périphériques :

TTY: *Télétype.*
CRT: *console écran/clavier.*
BAT: *batch (train de travaux) : lecteur de rubans ou de cartes perforés.*
UC1: *console non standard définie par l'utilisateur.*
UR1:, **UR2:** *lecteurs non standards définis par l'utilisateur.*
PTP: *perforateur de rubans.*
PTR: *lecteur de rubans.*
UP1:, **UP2:** *perforateurs non standards définis par l'utilisateur.*
LPT: *imprimante.*
UL1: *sortie de listes non standard définie par l'utilisateur.*

Ces appellations sont liées aux organes périphériques de l'ordinateur **Intellec 800** d'Intel, l'une des premières machines sur laquelle le CP/M a fait ses débuts.

A l'heure actuelle, il est rare de rencontrer un lecteur/perforateur de rubans sur nos ordinateurs modernes. Toutefois, l'unité de cassette magnétique (cassettophone) par exemple, peut lui être substituée à condition que le BIOS ait été écrit en conséquence.

La forme de la commande suivante est utilisée pour connaître les informations relatives à un fichier donné :

```
A>STAT MOVCPM.COM

 Recs  Bytes  Ext Acc
   76    10k    1 R/W A:MOVCPM.COM
Bytes Remaining On A: 156k

A>
```

Nous savons ainsi que le fichier MOVCPM.COM occupe 76 enregistrements (record = secteur de 128 octets dans notre cas) sur le disque, que la longueur allouée à ce fichier est de 10K octets, qu'un seul bloc d'allocation lui est réservé (extent) (voir appendices) et qu'il n'est pas protégé contre l'écriture (R/W = Read/Write). Cela peut être différent sur votre système.

Le paramètre **$S** peut être ajouté à la suite de la commande précédente :

```
STAT MOVCPM.COM $S

 Size  Recs  Bytes  Ext Acc
   76    76    10k    1 R/W A:MOVCPM.COM
Bytes Remaining On A: 156k

A>
```

ce qui provoque l'impression d'un champ supplémentaire (Size), utilisé en gestion de fichiers à accès direct. En accès séquentiel, sa valeur est identique à celle du champ Rec.

De la même façon, nous pouvons lister les caractéristiques d'un groupe de fichiers :

```
A>STAT *.ASM

 Recs  Bytes  Ext Acc
   96    12k    1 R/W A:BIOS.ASM
   69     9k    1 R/W A:CBIOS.ASM
   33     5k    1 R/W A:DUMP.ASM
Bytes Remaining On A: 156k

A>
```

et même de la totalité des fichiers présents sur la disquette :

```
A>STAT *.*

 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASM.COM
   96    12k    1 R/W A:BIOS.ASM
   69     9k    1 R/W A:CBIOS.ASM
   38     5k    1 R/W A:DDT.COM
   33     5k    1 R/W A:DUMP.ASM
    4     1k    1 R/W A:DUMP.COM
   60     8k    1 R/W A:DUMP.PRN
   52     7k    1 R/W A:ED.COM
   14     2k    1 R/W A:LOAD.COM
   76    10k    1 R/W A:MOVCPM.COM
   58     8k    1 R/W A:PIP.COM
   41     6k    1 R/W A:STAT.COM
   10     2k    1 R/W A:SUBMIT.COM
    8     1k    1 R/W A:SYSGEN.COM
    6     1k    1 R/W A:XSUB.COM
Bytes Remaining On A: 156k

A>
```

Mais il peut être intéressant de "masquer" certains fichiers lors de l'édition du catalogue. Ces fichiers existent toujours, bien entendu, mais ils n'apparaitront plus dans le catalogue. Cela s'appelle *donner un attribut d'invisibilité à un fichier*.

Cette opération peut s'effectuer par un paramètre de la commande **STAT** :

```
A>STAT *.ASM $SYS

DUMP.ASM set to SYS
BIOS.ASM set to SYS
CBIOS.ASM set to SYS
A>
```

Lorsque le mot-clé "**$SYS**" figure dans la commande, le ou les fichiers nommés sont rendus invisibles (SYStem). La preuve ?

```
A>DIR
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     PRN
A>
```

Tous les fichiers ayant pour type "**ASM**" ne figurent plus dans la liste du catalogue.

De manière analogue, il est également possible de protéger un ou plusieurs fichiers contre une écriture intempestive, souvent accidentelle :

```
A>STAT S?????.* $R/O

SUBMIT.COM set to R/O
STAT.COM set to R/O
SYSGEN.COM set to R/O
A>
```

Dans l'exemple ci-dessus, les fichiers dont le nom commence par la lettre "**S**" et possédant de 1 à 6 caractères se trouvent protégés contre toute opération d'écriture, tentative de suppression ou même de changement de nom. Ils prennent l'attribut R/O (Read Only = Lecture seulement).

```
A>stat *.*

 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASM.COM
   96    12k    1 R/W A:(BIOS.ASM)
   69     9k    1 R/W A:(CBIOS.ASM)
   38     5k    1 R/W A:DDT.COM
   33     5k    1 R/W A:(DUMP.ASM)
    4     1k    1 R/W A:DUMP.COM
   60     8k    1 R/W A:DUMP.PRN
   52     7k    1 R/W A:ED.COM
   14     2k    1 R/W A:LOAD.COM
   76    10k    1 R/W A:MOVCPM.COM
   58     8k    1 R/W A:PIP.COM
   41     6k    1 R/O A:STAT.COM
   10     2k    1 R/O A:SUBMIT.COM
    8     1k    1 R/O A:SYSGEN.COM
    6     1k    1 R/W A:XSUB.COM
Bytes Remaining On A: 156k

A>
```

La liste ci-dessus nous montre bien le changement d'attribut opéré sur ces fichiers. Notez que les fichiers rendus invisibles précédemment apparaissent entre parenthèses.

Pour rendre les fichiers visibles de nouveau, il suffit d'exécuter la commande :

```
A>STAT *.ASM $DIR

DUMP.ASM set to DIR
BIOS.ASM set to DIR
CBIOS.ASM set to DIR
A>
```

(**$DIR** signifiant que les fichiers seront visibles dans la DIRectory). Nous retrouvons alors notre liste d'origine :

```
A>DIR
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : DUMP     PRN
A>
```

S'il s'avère nécessaire d'effectuer une opération d'écriture sur un fichier ayant l'attribut R/O, on peut exécuter la commande :

```
A>STAT S?????.* $R/W

SUBMIT.COM set to R/W
STAT.COM set to R/W
SYSGEN.COM set to R/W
A>
```

en plaçant le mot-clé **$R/W** de la même façon que nous avions placé précédemment le mot-clé **$R/O.**

Il peut être utile, d'autre part, de protéger de manière temporaire, la totalité d'une disquette (par exemple avant une manipulation hasardeuse !). Cela se fait tout simplement par :

```
A>STAT B:=R/O

A>
```

Dans notre cas, toute opération d'écriture sur la disquette B: sera vouée à l'échec. La commande suivante nous indique bien que B: est protégée :

```
A>STAT
A: R/W, Space: 156k
B: R/O, Space: 241k
```

Il faut noter que cet état n'est pas conservé sur la disquette (comme il l'est pour la protection d'un fichier), *mais en mémoire uniquement.*

Pour revenir à l'état initial, il est nécessaire d'effectuer au minimum un démarrage à chaud :

```
A>^C
A>STAT
A: R/W, Space: 156k

A>STAT B:

Bytes Remaining On B: 241k

A>STAT
A: R/W, Space: 156k
B: R/W, Space: 241k

A>
```

C comme Chaud !

Mais la commande STAT, qui réalise bien des choses, comme nous pouvons le constater jusqu'ici, permet également de modifier l'affectation des organes périphériques. Nous avons vu précédemment que la sortie de listes (LST:) était affectée à la Télétype (TTY:). Pour effectuer l'impression des listes sur une véritable imprimante, il suffira d'écrire :

```
A>STAT LST:=LPT:

A>
```

Cela sous-entend que le BIOS de notre CP/M est capable de "conduire" l'imprimante (LPT:) raccordée au système. La nouvelle affectation sera maintenant :

```
A>STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:
```

De même que pour la protection temporaire R/O du disque, ces affectations résident uniquement en mémoire. Elles sont fixées à la génération du système mais peuvent toutefois être modifiées (voir appendices).

Pour revenir à l'état initial, un démarrage à chaud :

```
A>^C
A>STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:

A>
```

ne suffit pas... ! Il est nécessaire de re-initialiser entièrement le système.

EN RESUME

```
+-------------------------------------------------------------+
!  STAT - commande de listase des informations sur les        !
          disques et les programmes                           !
!       - modification des affectations peripheriques          !
+-----------------------------+-------------------------+
!          SYNTAXE            !        EXEMPLES         !
+-----------------------------+-------------------------+
!  STAT <>                    !  STAT                   !
!       DEV:                  !  STAT DEV:              !
!       USR:                  !  STAT USR:              !
!       VAL:                  !  STAT VAL:              !
!       DSK:                  !  STAT DSK:              !
!       <d:>                  !  STAT B:                !
!           =R/O              !  STAT B:=R/O            !
!           DSK:              !  STAT A:DSK:            !
!       <ref ambisue ou       !  STAT D??P.A??          !
!            explicite>       !  STAT PROG.COM          !
```

SYNTAXE	EXEMPLES
$SYS	STAT ED.COM $SYS
$DIR	STAT B:PROG.* $DIR
$R/O	STAT PROG.COM $R/O
$R/W	STAT *.PRO $R/W
$S	STAT *.* $S
commande transitoire - etendue en version 2	
voir complement ---> PIP, SAVE	

REN - commande de changement de nom de fichier

Il est souvent souhaitable de *changer le nom d'un programme* ou même - pourquoi pas - *d'une commande transitoire* figurant sur la disquette.

Tous les systèmes d'exploitation sont munis de cette commande. CP/M en comporte une qui se nomme **REN** (REName = renommer).

Il faudra s'habituer à la syntaxe qui est l'inverse de celle rencontrée sur les autres systèmes d'exploitation. En assembleur, l'instruction :

```
MOV A,B
```

exécute un transfert entre deux registres. A désigne le registre **destination** et B le registre **source.**

CP/M adopte le même principe, respectant ainsi l'homogénéité des règles syntaxiques :

```
REN <nouveau nom>=<ancien nom>
```

Ainsi, si le nom de l'éditeur **ED** n'évoque rien pour nous, nous pouvons le baptiser EDITEUR :

```
A>REN EDITEUR.COM=ED.COM
A>
```

et il figurera désormais ainsi dans le catalogue :

```
A>DIR
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: EDITEUR  COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : DUMP     PRN
A>
```

Un programme peut être protégé contre le changement de nom s'il possède un attribut adéquat (R/O). Vérifions-le sans plus attendre :

```
A>STAT EDITEUR.COM $R/O

EDITEUR.COM set to R/O
A>REN ED.COM=EDITEUR.COM
Bdos Err On A: File R/O
A>STAT EDITEUR.COM $R/W

EDITEUR.COM set to R/W
A>REN ED.COM=EDITEUR.COM
A>
```

Bien fait !

Notez l'erreur provoquée par la tentative de changement de nom d'un fichier R/O.

EN RESUME

REN - commande de changement de nom de fichier	
SYNTAXE	EXEMPLES
REN <nouveau nom>= <ancien nom>	REN NOUV.COM=ANC.COM REN B:PROG2=PROG1
commande résidente	

TYPE - commande d'affichage d'un fichier ASCII

La commande suivante va nous permettre d'*examiner le contenu d'un fichier*, pourvu que celui-ci soit composé de caractères affichables, c'est-à-dire ASCII.

C'est le cas pour le fichier DUMP.ASM figurant sur notre disquette, qui est le code source en assembleur du programme **DUMP.**

```
A>TYPE DUMP.ASM
;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF
A>
```

↑C

Durant l'édition, il est possible de stopper momentanément l'affichage en appuyant sur les touches **CTRL** et **S.** Pour stopper définitivement la liste, (ce que nous avons fait, étant donnée la longueur du programme), il suffit de frapper sur **CTRL** et **C.**

EN RESUME

TYPE - commande d'affichage d'un fichier ASCII	
SYNTAXE	EXEMPLES
TYPE <ref explicite>	TYPE PROG.ASM TYPE B:PROG2.ASM
commande residente	

PIP - utilitaire de copie fichiers/périphériques

Abordons maintenant une commande "utilitaire" qui réside sur disquette et qui *réalise toutes les fonctions d'échanges entre les divers organes du système :* **PIP** (Peripheral Interchange Program = programme d'échange entre périphériques).

Etant donnée l'étendue des possibilités de cette commande, nous examinerons **PIP** plus profondément au chapitre suivant, en nous contentant pour l'instant de ses aspects les plus usuels.

L'une des fonctions les plus courantes est la *copie de programmes*. Vous désirez par exemple, effectuer des modifications sur un programme, mais vous souhaitez conserver l'original intact. Très simple !

```
A>PIP PROG=DUMP.COM

A>
```

Notez que la syntaxe est la même que celle utilisée avec **REN** :

PIP <nouvelle copie>=<programme d'origine>

Lorsque la copie s'effectue sur la même disquette, il est nécessaire de donner un nom différent au fichier ainsi créé, faute de quoi l'opération sera nulle. Une liste du catalogue nous confirme bien qu'un NOUVEAU programme figure maintenant sur la disquette (c'est notre première création) sous le nom PROG :

```
A>DIR
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : PROG         : DUMP     PRN
A>
```

En pratique, il est plus fréquent de copier un programme d'une unité à l'autre, comme ceci :

```
A>PIP B:=DUMP.COM

A>PIP B:EDITEUR.COM=ED.COM
```

DUMP. COM figure maintenant sur la disquette B:, ainsi que l'éditeur ED. Notez que ce dernier exemple réalise du même coup un changement de nom, ce qui correspond à :

PIP B:=ED.COM

suivi de

```
REN B:EDITEUR.COM=ED.COM
```

Bien entendu, il s'agit d'une *duplication* : les programmes d'origine figurent toujours sur la disquette A:.

Pour copier la totalité des programmes ".COM" par exemple, sur la disquette B:, il faudra faire :

```
A>PIP B:=*.COM

COPYING -
MOVCPM.COM
PIP.COM
SUBMIT.COM
XSUB.COM
ED.COM
ASM.COM
DDT.COM
LOAD.COM
STAT.COM
SYSGEN.COM
DUMP.COM

A>
```

Nous pouvons constater que CP/M ne nous laisse pas dans l'ignorance de ses actes puisqu'il liste les programmes au fur et à mesure qu'il les copie.

La liste du catalogue de B: nous donne maintenant :

```
A>DIR B:
B: EDITEUR  COM : MOVCPM   COM : PIP      COM : SUBMIT   COM
B: XSUB     COM : ED       COM : ASM      COM : DDT      COM
B: LOAD     COM : STAT     COM : SYSGEN   COM : DUMP     COM
A>
```

Pour copier tous les programmes d'une disquette, vous l'avez sûrement deviné, il suffit de faire **PIP B:*.***

Comme nous l'avons signalé auparavant, **PIP** exécute des échanges entre les organes périphériques du système. Précédemment, nous avons examiné sur l'écran vidéo le contenu du fichier DUMP.ASM (en code ASCII). Que penseriez-vous d'une copie de ce fichier sur l'imprimante, par exemple ?

```
A>PIP LST:=DUMP.ASM

A>
```

Ce sera le cas si vous n'avez pas ré-initialisé le système depuis l'affectation des sorties de listes (LST:) à l'imprimante (LPT:). Le code de contrôle **CTRL P** aurait donné le même résultat, avec en plus, tous les dialogues avec la console (voir chapitre précédent).

Mais voici venue l'occasion rêvée de *créer notre propre programme*, en demandant à **PIP** de transférer les codes issus du clavier de la console vers un fichier disque que nous nommerons TOTO.ASC (pourquoi pas ?) :

```
A>PIP TOTO.ASC=CON:
```

Après la frappe de cette commande, nous nous apercevons qu'il ne se passe rien... ce qui est bon signe car nous avons maintenant "la main" au clavier de la console !

Alors allons-y.

```
Au clair de la lune, mon ami Pierrot....
A>
```

Le chapitre précédent nous a appris que le code **CTRL Z** était utilisé pour signaler la fin d'un fichier. Employons-le donc maintenant pour terminer notre message.

Le contrôle est rendu à CP/M. Pour en savoir plus :

```
A>dir
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : PROG         : TOTO     ASC
A: DUMP     PRN
A>
```

Et nous constatons que notre fichier figure maintenant sur le catalogue de la disquette ! Nous savons par ailleurs, lister le contenu d'un fichier.

```
A>TYPE TOTO.ASC
Au clair de la lune, mon ami Pierrot....
```

Magnifique !

Nous pouvons aussi essayer d'envoyer un texte directement sur l'imprimante.

```
A>PIP LST:=CON:
Au clair de la lune, mon ami Pierrot
prete-moi ta plume...
A>
```

en n'oubliant pas de le terminer par CTRL Z.

Il est possible d'appeler le programme **PIP** sans lui fournir nécessairement des paramètres. Dans ce cas, le symbole "*" est affiché, ce qui signifie que **PIP** attend votre commande. Il suffit alors de frapper les paramètres concernés. Il est ainsi possible de réaliser différentes opérations sans qu'il soit nécessaire de charger le programme à chaque fois.

Vous avez maintenant quelques éléments vous permettant de réaliser pas mal de petites choses. Entraînez-vous !

EN RESUME

PIP - utilitaire de copie fichiers/periphériques	
SYNTAXE	**EXEMPLES**
PIP <> <dest>=<source>	PIP PIP COP=ORIG PIP B:=*.C?M PIP B:COP=ORIG PIP LST:=PROG.ASM PIP LST:=*.ASM PIP PROG.TXT=CON: PIP LST:=CON:
dest est une référence explicite ou un organe de sortie. source est une référence explicite ou ambigue, ou un organe d'entrée.	
commande transitoire	
voir complément à SAVE et au chapitre 4	

d: - affectation de la disquette implicite

Maintenant que notre seconde disquette (B:) contient quelques programmes, il est temps de signaler une commande résidente du CP/M qui permet de *changer l'affectation de la disquette implicite*.

Expliquons-nous. En frappant la commande STAT, par exemple, le système cherche le programme STAT.COM sur la disquette A:, sans qu'il soit nécessaire de préciser que cette commande se trouve effectivement sur A:. A: est la disquette implicite affectée au système.

Si STAT.COM n'y figure pas, une erreur est signalée et les choses s'arrêtent là.

Certains systèmes d'exploitation iraient consulter le catalogue de la disquette suivante avant de signaler l'erreur. CP/M ne sait pas faire cela.

Dans l'hypothèse où STAT.COM est présent sur la disquette B:, nous pouvons charger cette commande en frappant :

```
B:STAT
```

Dans le cas où ce genre d'opérations se renouvelle fréquemment, il peut être pratique de changer l'affectation de la disquette implicite. *Cela s'effectue tout simplement en frappant le nom de cette disquette :*

```
A>B:
B>
```

Remarquez que "B>" remplace le traditionnel "A>" en début de ligne, indiquant que CP/M attend une commande.

En frappant la commande (résidente) DIR, c'est le catalogue de B: qui est listé :

```
B>DIR
B: EDITEUR  COM : MOVCPM   COM : PIP      COM : SUBMIT   COM
B: XSUB     COM : ED       COM : ASM      COM : DDT      COM
B: LOAD     COM : STAT     COM : SYSGEN   COM : DUMP     COM
B>
```

Cette fois-ci, la commande (transitoire) STAT sera cherchée sur la disquette B: :

```
B>STAT
A: R/W, Space: 154k
B: R/W, Space: 183k
```

Pour revenir à l'état initial (disquette implicite A:), il suffira bien évidemment de frapper :

```
B>A:
A>
```

EN RESUME

d: - affectation de la disquette implicite	
SYNTAXE	EXEMPLES
<d:>	B: A:
commande residente	

ERA - commande de destruction d'un fichier

Il est fréquent que certains programmes aient une "existence" passagère. Afin d'éviter qu'ils encombrent inutilement l'espace disque, il est nécessaire de pouvoir disposer d'une *commande de destruction de fichiers*.

Celle du CP/M répond au doux nom d'**ERA** (ERAze = effacement).

```
A>ERA PROG
A>
```

La commande ci-dessus libère sur la disquette l'espace occupé par le programme PROG. Son nom disparaît également du catalogue.

Cette commande dangereuse peut également supprimer la totalité des programmes d'une disquette :

```
A>ERA B:*.*
```

Flairant la méprise, CP/M nous demande toutefois de confirmer notre ordre.

```
ALL (Y/N)?
```

Ce à quoi nous répondons "Y" (Yes) :

```
                Y
Bdos Err On B: File R/O
A>
```

Et malin avec ça...!

Tout compte fait, elle n'est pas si dangereuse que cela ! Elle refuse fort heureusement de supprimer des programmes protégés !

```
A>STAT B:*.* $R/W

MOVCPM.COM set to R/W
PIP.COM set to R/W
SUBMIT.COM set to R/W
XSUB.COM set to R/W
ED.COM set to R/W
ASM.COM set to R/W
DDT.COM set to R/W
LOAD.COM set to R/W
STAT.COM set to R/W
SYSGEN.COM set to R/W
DUMP.COM set to R/W
A>
```

```
A>ERA B:*.*
ALL (Y/N)?Y
A>
```

Maintenant, il n'est plus possible de revenir en arrière, ainsi que nous le certifie la commande ci-dessous :

```
A>DIR B:
NO FILE
A>
```

En réalité, nous l'avons bien cherché !

EN RESUME

ERA - commande de destruction d'un fichier	
SYNTAXE	EXEMPLES
ERA <ref explicite ou ref ambigue>	ERA PROG ERA *.BAK ERA B:*.*
commande residente	

DUMP - commande d'affichage d'un fichier en hexadécimal

Nous avons vu précédemment comment lister un fichier ASCII sur l'écran ou sur l'imprimante, grâce à la commande TYPE. Mais qu'en est-il d'un fichier binaire, dont les codes ne sont pas affichables ?

*La commande **DUMP** nous permet de lister en code hexadécimal le contenu d'un fichier de la disquette.*

Le programme source de cet utilitaire est d'ailleurs fourni avec le CP/M standard, nous donnant ainsi un excellent exemple de réalisation d'une commande système.

```
A>DUMP TOTO.ASC

0000 41 75 20 63 6C 61 69 72 20 64 65 20 6C 61 20 6C
0010 75 6E 65 2C 20 6D 6F 6E 20 61 6D 69 20 50 69 65
0020 72 72 6F 74 2E 2E 2E 2E 0D 0A 1A 1A 1A 1A 1A 1A
0030 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0040 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0050 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0060 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0070 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A

A>
```

Le texte que nous avions introduit plus haut, au clavier, grâce à la commande **PIP**, apparaît maintenant en hexadécimal. La liste est toujours multiple d'un secteur (128 octets), ce qui explique les "1A" qui complètent notre texte et qui correspondent au code CTRL Z marquant la fin du fichier.

EN RESUME

DUMP - commande d'affichage d'un fichier en hexa	
SYNTAXE	EXEMPLES
DUMP <ref explicite>	DUMP PROG.COM DUMP B:PROG.BIN
commande transitoire	

ED - programme de création et d'édition de fichiers

La manière dont nous avons créé notre premier fichier était simple mais peu évoluée, il faut le reconnaître : en cas d'erreur, il ne restait plus qu'à tout refrapper depuis le début !

Nous allons maintenant examiner quelques commandes simples offertes par l'**éditeur de textes de CP/M**, qui nous donnera une plus grande souplesse dans la création de nos fichiers ou de nos programmes.

La réalisation de cet éditeur appelé **ED**, remonte à l'époque de la Télétype pour laquelle il a d'ailleurs été écrit. Cela signifie qu'il est particulièrement mal adapté aux consoles à écran cathodique qui sont raccordées à nos ordinateurs modernes (éditeur ligne opposé à éditeur écran), ce qui sous-entend l'absence de possibilité de déplacement d'un curseur dans les quatre directions.

On peut s'étonner, étant donné le succès remporté par CP/M, que Digital Research n'ait pas réactualisé cet éditeur ancestral.

Nous voici donc ramenés près de 10 ans en arrière - mais avec le bruit et la lenteur en moins...!

A la décharge de cet éditeur, il faut toutefois souligner qu'il peut être simple et pratique d'emploi, si on se limite aux commandes les plus usuelles, celles que nous allons examiner ci-dessous. Les compléments concernant ED seront examinés dans le chapitre suivant.

Pour *créer un fichier* (que nous nommerons ESSAI.TXT par exemple) avec l'éditeur, il suffit de frapper :

```
ED ESSAI.TXT
```

Dans le cas présent, ce nom n'existe pas dans le catalogue de la disquette et l'éditeur nous le fait savoir en précisant qu'il s'agit d'un nouveau fichier (NEW FILE). Mais observons globalement la suite des évènements...

```
NEW FILE
     : *i
    1:  CP/M est un systeme d'exploitation
    2:  de disquette tres simple d'emploi...
    3:  Son editeur est pourvu de nombreuses
    4:  commandes qui lui donnent parfois une
    5:  apparence un peu complexe ! Pourtant,
    6:  quelques commandes simples suffisent
    7:  pour savoir le mettre en oeuvre.
    8:
     : *e
A>
```

Après le symbole "*" affiché par l'éditeur et qui indique le niveau "acceptation de commande", frappons "**i**" (i comme introduction ou *insertion*), suivi d'un code "retour chariot". **ED** répond par un numéro de ligne, puis attend notre message. A chaque "retour chariot", le numéro de ligne suivant apparaît (pointeur ligne), et ainsi de suite jusqu'à la fin du texte (ligne 8). Parvenu à cet endroit, il suffit de frapper le code **CTRL Z** *pour terminer l'insertion*. **ED** affiche de nouveau un "*", signifiant par là qu'il attend une autre commande. Frappons "**e**" (suivi de "retour chariot") pour indiquer à ED que nous voulons maintenant *terminer l'opération d'édition* (e comme end = fin, ou exit = sortie). Le fichier est alors sauvé sur disquette et le contrôle est rendu à CP/M.

En résumé, pour créer un fichier :

- "**i**" (commande insertion)
- introduction du texte
- **CTRL Z** (fin de texte)
- "**e**" (commande fin)

Et si vous avez le moindre doute sur la réalité du fichier ainsi créé, n'hésitez pas à frapper :

```
A>type essai.txt
CP/M est un systeme d'exploitation
de disquette tres simple d'emploi...
Son editeur est pourvu de nombreuses
commandes qui lui donnent parfois une
apparence un peu complexe ! Pourtant,
quelques commandes simples suffisent
pour savoir le mettre en oeuvre.

A>
```

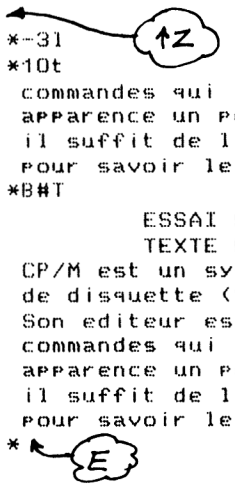
Remarque : Si le "**i**" de la commande d'insertion est frappé en majuscule ("**I**"), le texte sera sauvé en majuscule sur la disquette, bien qu'il apparaisse en minuscules sur l'écran.

Nous allons maintenant rappeler notre texte et lui faire subir quelques changements. Nous ajouterons tout d'abord un titre en tête du texte, quelques mots au milieu d'une phrase, et nous remplacerons un mot par un autre.

Voici le listing complet de ces opérations.

```
ed essai.txt
     : *i
    1:          ESSAI DE CREATION D'UN
    2:          TEXTE PAR L'EDITEUR ED
    3:
     : *#A                ↑Z
    3: *B
    1: *#T
    1:          ESSAI DE CREATION D'UN
    2:          TEXTE PAR L'EDITEUR ED
    3:  CP/M est un systeme d'exploitation
    4:  de disquette tres simple d'emploi...
    5:  Son editeur est pourvu de nombreuses
    6:  commandes qui lui donnent parfois une
    7:  apparence un peu complexe ! Pourtant,
    8:  quelques commandes simples suffisent
    9:  pour savoir le mettre en oeuvre.
    1: *fdisquette
    4: *t
 tres simple d'emploi...
    4: *i
    4:   (SED ou DOS)          ↑Z
    4: *b
    1: *  CR
    2:          TEXTE PAR L'EDITEUR ED
    2: *
    3:  CP/M est un systeme d'exploitation
    3: *
    4:  de disquette (SED ou DOS) tres simple d'emploi...
    4: *
    5:  Son editeur est pourvu de nombreuses
    5: *
    6:  commandes qui lui donnent parfois une
    6: *
    7:  apparence un peu complexe ! Pourtant,
    7: *scommandes^Zmanipulations
    8: *b
    1: *#t
    1:          ESSAI DE CREATION D'UN
    2:          TEXTE PAR L'EDITEUR ED
    3:  CP/M est un systeme d'exploitation
    4:  de disquette (SED ou DOS) tres simple d'emploi...
    5:  Son editeur est pourvu de nombreuses
    6:  commandes qui lui donnent parfois une
    7:  apparence un peu complexe ! Pourtant,
    8:  quelques manipulations simples suffisent
    9:  pour savoir le mettre en oeuvre.
    1: *7L
    8: *K
    8: *t
```

```
8:  pour savoir le mettre en oeuvre.
8: *i
8:  il suffit de l'experimenter quelques minutes
9:
9: *-31
6: *10t
6:  commandes qui lui donnent parfois une
7:  apparence un peu complexe ! Pourtant,
8:  il suffit de l'experimenter quelques minutes
9:  pour savoir le mettre en oeuvre.
6: *B#T
1:          ESSAI DE CREATION D'UN
2:          TEXTE PAR L'EDITEUR ED
3:  CP/M est un systeme d'exploitation
4:  de disquette (SED ou DOS) tres simple d'emploi...
5:  Son editeur est pourvu de nombreuses
6:  commandes qui lui donnent parfois une
7:  apparence un peu complexe ! Pourtant,
8:  il suffit de l'experimenter quelques minutes
9:  pour savoir le mettre en oeuvre.
1: *
```



Lorsque l'éditeur affiche l'astérisque, le fichier "essai.txt" n'est pas encore chargé en mémoire. Nous pouvons donc en profiter pour introduire notre titre (commande "**i**") en le terminant par **CTRL Z.** Le pointeur de ligne est alors positionné au début de la ligne 3.

Pour *appeler le fichier en mémoire, à la suite du texte précédemment introduit*, il suffit d'utiliser la commande **#A** ou **#a** (A comme Append = Ajouter). Le symbole **#**, pour l'éditeur, à le sens de "totalité" ou "plénitude". En réalité, il représente le plus grand nombre connu de l'éditeur : 65535.

Nous aurions pu frapper "**4A**" pour ajouter 4 lignes à la suite de notre titre. Un nombre supérieur au nombre de lignes du fichier aurait eu le même effet qu'un **#** (**100A** par exemple).

Après exécution de cette commande, le pointeur de ligne est toujours positionné en ligne 3. Pour le *placer au tout début du texte*, nous utiliserons la commande "**b**" ou "**B**" (B comme Begin = début). Le pointeur ligne passe alors en 1.

Pour afficher la totalité de notre texte, utilisons la commande "**t**" ou "**T**" (Type = affichage), précédée du signe **#**. L'éditeur liste alors le contenu du fichier. Remarquez que le pointeur de ligne reste sur la ligne 1. Pour afficher 3 lignes seulement, nous aurions pu faire "**3T**".

Nous allons maintenant *rechercher le mot* "disquette" dans le texte (le premier mot "disquette" rencontré).

Pour cela, frappons la commande "**f**" (f comme find = trouver) suivie du mot à chercher : **fdisquette.** Le pointeur de ligne s'arrête en ligne 4, *après le mot recherché*. En frappant la commande "**t**", nous voyons que seul, le texte suivant le mot "disquette" est affiché : "très simple d'emploi".

Si nous voulons introduire un texte à cet endroit, il suffit d'invoquer la commande d'insertion "**i**", d'entrer notre texte : "(SED ou DOS)", et de le terminer par **CTRL Z.**

La commande "**b**" nous amène au début du fichier. En frappant un "retour chariot", la ligne courante est affichée et le pointeur ligne passe à la ligne suivante. Après plusieurs tentatives, nous arrivons à la ligne 7.

Là, nous voulons *remplacer le mot* "commandes" se trouvant dans la suite du texte, par le mot "manipulations". La forme de cette commande est la suivante :

S<texte à remplacer>**CTRL Z**<nouveau texte>

Aussi, nous frappons : **scommandes** (**CTRL Z**) **manipulations.**

Nous retournons ensuite au début du fichier ("**b**") et nous listons notre texte ("**#t**").

Le pointeur ligne étant au début de la ligne 1, nous voulons maintenant *avancer ce pointeur de 7 lignes* (par exemple). Cela se fait par la commande "**l**" ou "**L**" (L comme ligne) : **7L.** Le pointeur affiche alors le numéro 8 (7 plus 1 égale 8).

La ligne 8 (sur laquelle se trouve le pointeur) ne nous plait plus, et nous voulons la supprimer. En frappant la commande "**k**" ou "**K**" (k comme kill = tuer), *l'éditeur supprime le texte de cette ligne.* Le pointeur de ligne affiche toujours le numéro 8, mais la ligne 8 contient maintenant le texte de l'ancienne ligne 9. Pour le constater, il suffit de frapper "**t**". En fait, tout le texte suivant le pointeur a subi une compression vers le haut, de la valeur d'une ligne.

Nous voulons ensuite remplacer l'ancienne ligne 8 (supprimée maintenant) par un autre texte. Cela se fait par la commande d'insertion ("**i**") suivie du texte terminé par **CTRL Z.** Après cette opération, le texte suivant le pointeur a subi une dilatation vers le bas, de la valeur d'une ligne.

Pour "remonter" le pointeur de 3 lignes, il suffit d'utiliser la commande "**l**" déjà vue plus haut, en la faisant précéder d'une valeur négative... : "**-3l**". Le pointeur passe alors en ligne 6 (9 moins 3 égale 6).

"**10t**" affiche alors le texte à partir de la ligne 6 jusqu'à la dernière ligne (il reste moins de 10 lignes).

Il est aussi possible d'*enchaîner plusieurs commandes*, comme par exemple : **B#T** qui va provoquer le déplacement du pointeur au début du fichier ("**B**") puis en lister le contenu ("**#T**").

Si le texte vous plait ainsi, n'oubliez pas la commande "**E**" pour le sauver sur disquette. Dans le cas contraire, frappez la commande "**q**" : vous quitterez l'éditeur sans toucher à votre fichier original.

Si vous avez la curiosité de lister le catalogue de la disquette après ces modifications, vous vous apercevrez qu'un fichier "essai.bak" y figure... C'est une sage précaution de la part de l'éditeur qui, semble t-il, n'a aucune confiance en nous et qui *conserve toujours une version du fichier édité avant modifications* en lui donnant le type BAK (de backup = copie de sauvegarde).

Ainsi, si vous préférez après réflexion, en rester à l'ancienne version, il suffira de faire :

```
ERA ESSAI.TXT
REN ESSAI.TXT=ESSAI.BAK
```

Les quelques manipulations que nous avons examinées avec l'éditeur **ED**, devraient vous permettre de créer et de modifier n'importe quel fichier avec un minimum de tourments.

Il vous faut toutefois pratiquer afin de bien vous imprégner de ces différentes commandes, en vous aidant du tableau ci-dessous. Cela, aucun livre ne peut le faire à votre place !

EN RESUME

ED - programme de creation et d'edition de fichiers	
SYNTAXE	EXEMPLE
ED <ref explicite>	ED ESSAI.TXT
commande transitoire	

Commandes usuelles ED	Fonctions realisees
I	- Insertion jusqu'a CTRL Z
A (nA , #A)	- Ajoute lignes du fichier disque en memoire
E	- fin. Sortie de l'editeur avec mise a jour fichier
Q	- Quitte l'editeur sans mise a jour
B	- retour au deBut fichier
T (nT , #T)	- affichage ligne de Texte
L (nL)	- avance Ligne (+ ou -)
K (nK)	- suppression ligne
<retour chariot>	- avance ligne + affichage
F<texte>	- recherche <texte>
S<ancien>↑Z<nouveau>	- Substituer <ancien> texte par <nouveau> texte

SUBMIT - commande d'enchaînement

Nous allons maintenant voir comment CP/M peut *exécuter, de façon automatique, une suite d'opérations pré-définies*, en utilisant la commande **SUBMIT** (Soumission d'une tâche).

Le principe est simple : il faut créer un fichier (appelé tout naturellement "fichier de commandes") et le "soumettre" pour exécution à CP/M, via la commande **SUBMIT.** Ce fichier décrira la suite des différentes commandes à exécuter, de la même façon qu'elles seraient frappées au clavier de la console.

Nous savons créer un fichier par **ED**, n'est-ce pas ?

```
A>ed chain.sub

NEW FILE
     : *i
    1:  dir *.asm
    2:  stat
    3:  ↑Z
     : *e
A>
```

Ceci étant fait, nous nous proposons ici d'enchaîner deux opérations : **DIR *.ASM** (liste du catalogue des fichiers ".asm"), et **STAT** (état de la disquette). Notez que le fichier "chain" porte le type ".sub", ce qui est nécessaire pour la suite des opérations.

Lançons maintenant l'exécution de cette tâche :

```
A>submit chain

A>DIR *.ASM
A: DUMP      ASM : BIOS      ASM : CBIOS     ASM
A>STAT
A: R/W, Space: 151k


A>
```

Remarquez que **SUBMIT** attend le type implicite ".sub" qu'il ne faut donc pas écrire dans la commande. Après exécution, **SUBMIT** rend le contrôle à CP/M.

Cette commande est bien pratique, mais elle semble d'un usage limité. Ne serait-il pas possible de "paramètrer" son déroulement ?

La réponse est heureusement positive ! Il est en effet possible, au moment de la "soumission", de passer des paramètres à la commande **SUBMIT.** Dans le fichier de commandes, ces "variables" seront nommées par le symbole "**$**" suivi d'un numéro dépendant du rang du paramètre considéré : **$1** pour le premier paramètre, **$2** pour le second, etc..., auxquelles seront substitués les vrais paramètres au moment de la soumission :

SUBMIT <nom fichier de commande> <$1> <$2>...

Fabriquons donc un nouveau fichier par la méthode qui nous est devenue familière maintenant :

```
A>ed bof.sub

NEW FILE
     : *i
    1:  stat $1
    2:  ren $2.pro=$1
    3:  stat $2.pro
    4:  ren $1=$2.pro
    5:
     : *e

A>
```

↑Z

La tâche réalisée dans ce fichier de commandes appelle quelques explications. Disons tout de suite qu'elle ne présente qu'un intérêt d'ordre pédagogique.

Voici ce que nous allons faire : vérifier qu'un fichier de nom donné (paramètre $1) est présent sur la disquette ; changer le nom de ce fichier en le remplaçant par un second nom (donné par le paramètre $2) ; vérifier que le fichier apparaît cette fois sous ce nouveau nom ; restituer au fichier son nom d'origine.

La forme de la commande sera donc :

SUBMIT BOF <ancien nom> <nouveau nom>

et son exécution donne ceci :

```
A>submit bof essai.txt texte

A>STAT ESSAI.TXT

 Recs  Bytes  Ext Acc
    3     1k    1 R/W A:ESSAI.TXT
Bytes Remaining On A: 150k

A>REN TEXTE.PRO=ESSAI.TXT
A>STAT TEXTE.PRO

 Recs  Bytes  Ext Acc
    3     1k    1 R/W A:TEXTE.PRO
Bytes Remaining On A: 150k

A>REN ESSAI.TXT=TEXTE.PRO
A>
```

Remarquez, au moment de l'exécution, comment CP/M remplace **$1** par **ESSAI.TXT** et **$2** par **TEXTE...**

Expérimentez avec d'autres exemples de votre composition.

EN RESUME

SUBMIT - commande d'enchainement	
SYNTAXE	EXEMPLES
SUBMIT <nom fichier de commande> <$1> <$2> ...	SUBMIT CHAIN SUBMIT DO PROG A * C
commande transitoire	
voir extension (version 2) ---> XSUB	

ASM - programme assembleur

CP/M est un système d'exploitation complet, qui se suffit à lui-même. Ainsi, la *version standard comprend* un éditeur de textes et *un assembleur*. Nous allons maintenant apprendre à manipuler ce dernier.

A l'origine, CP/M a été conçu sur un ordinateur équipé d'un microprocesseur 8080 d'Intel. Il n'est donc pas étonnant que l'assembleur **ASM** (c'est son nom) fourni en standard sur la disquette CP/M soit conçu pour les instructions de ce microprocesseur (mnémoniques Intel).

Il ne sera pas question ici, de nous étendre sur le langage assembleur, d'autres ouvrages y étant consacrés (*).

* du même auteur "Programmer en assembleur" aux Editions du P.S.I.

Nous partirons donc du principe que le lecteur est déjà familiarisé avec celui-ci et nous insisterons surtout sur sa mise en oeuvre ainsi que sur les détails qui lui sont spécifiques (voir aussi les appendices).

Les trois phases d'une opération d'assemblage sont les suivantes :

- création d'un programme source par l'éditeur (**.ASM**),
- assemblage de ce programme avec génération d'un programme objet (**.HEX**) et d'une liste d'assemblage (**.PRN**),
- création à partir du programme objet, d'un programme exécutable par CP/M (**.COM**).

La première phase est réalisée par **ED**, et nous savons maintenant comment nous y prendre pour cela. La seconde phase est assumée par **ASM**. Quant à la troisième, c'est la commande **LOAD** que nous examinerons bientôt, qui s'en chargera.

Le lancement de l'assembleur s'effectue par une commande de la forme :

ASM <nom fichier source>.<S><O><L>

Le fichier source a obligatoirement pour type "**.ASM**", mais celui-ci ne figure pas dans la commande. A sa place, on lui substitue trois paramètres ayant la signification suivante :

```
                                                  <S><O><L>
                                                   !  !  !
+-----------------------------------+              !  !  !
!  A a P : nom disquette contenant   <------+      !  !  !
!          le fichier Source         !             !  !  !
+-----------------------------------+             !  !  !
                                                  !  !  !
+-----------------------------------+             !  !  !
!  A a P : nom disquette devant con- <---------+   !  !  !
!          tenir le fichier objet    !             !  !  !
!  Z     : saut generation du        !             !  !  !
!          fichier Objet             !             !  !  !
+-----------------------------------+             !  !  !
                                                  !  !  !
+-----------------------------------+             !  !  !
!  A a P : nom disquette devant con- <-------------+  !  !
!          tenir le fichier Liste    !             !  !  !
!  X     : sortie liste a la console !             !  !  !
!  Z     : saut generation de liste  !             !  !  !
+-----------------------------------+             !  !  !
```

Le fichier objet génèré par l'assembleur portera le type implicite "**.HEX**" ; le fichier liste quant à lui, portera le type "**.PRN**".

Dans le cas où les paramètres <SOL> sont absents, "**.AAA**" est pris par défaut.

Voici deux exemples d'appel de l'assembleur :

ASM PROG.ABX	source : PROG.ASM, sur A: objet : PROG.HEX placé sur B: listing généré à la console
ASM ESSAI.AZA	source : ESSAI.ASM sur A: pas de fichier objet généré fichier liste : ESSAI.PRN placé sur A:

Mais, avant de pouvoir assembler un programme, il faut tout d'abord en créer un, les fichiers sources (ASM) fournis sur la disquette CP/M n'étant exploitables qu'avec le macro-assembleur (non fourni en standard).

Nous avons choisi pour cet exemple, un programme très simple utilisant deux appels de routines du CP/M (lecture clavier et affichage vidéo).

La tâche de ce programme consistera à saisir une ligne au clavier puis à l'afficher sur l'écran. La sortie du programme (retour sous contrôle du CP/M) s'effectuera en entrant le caractère "#". Il est donc difficile de trouver plus simple !

Créons d'abord le programme, nous en discuterons après.

```
A>ed prog.asm

NEW FILE
   : *i
    1:          org     100h      ; debut TPA
    2:  ;
    3:  exit    equ     0         ; boot a chaud
    4:  bdos    equ     5         ; point entree BDOS
    5:  input   equ     10        ; routine entree ligne
    6:  print   equ     9         ; routine sortie ligne
    7:  ;
    8:  prog    lxi     d,bufrec          ; bloc commande entree
    9:          mvi     c,input           ; code entree
   10:          call    bdos              ; appel bdos
   11:  ;
   12:          call    ligne             ; envoi cr/lf
   13:  ;
   14:          lda     texte             ; premier caractere texte
   15:          cpi     '#'               ; est-ce # ?
   16:          jz      exit              ; oui
   17:  ;
   18:          lxi     h,texte           ; debut texte
   19:          lda     ncr               ; nbre carac recus
   20:          mov     e,a               ; dans registre E
   21:          mvi     d,0               ; reg D =0
   22:          dad     d                 ; (HL)+(DE)-->HL
   23:          mvi     m,'$'             ; fin texte par $
   24:  ;
   25:          lxi     d,texte           ; pointe debut texte
   26:          mvi     c,print           ; code sortie
   27:          call    bdos              ; appel bdos
   28:  ;
   29:          call    ligne             ; envoi cr/lf
   30:  ;
   31:          xra     a                 ; 0--> nbre carac recus
   32:          sta     ncr               ;
   33:          jmp     prog              ; suite
   34:  ;
   35:  ligne   lxi     d,crlf            ; routine envoi cr/lf
   36:          mvi     c,print           ; code sortie
   37:          jmp     bdos              ; apeel bdos + retour
   38:  ;
   39:  ; zones des donnees ------------
   40:  ;
   41:  bufrec  db      50        ; longueur buffer
   42:  ncr     db      0         ; nbre carac recus
   43:  texte   ds      51        ; buffer texte
   44:  crlf    db      0dh,0ah,'$'
   45:          end     prog
   46:
   : *e

A>
```

GRRRR!

↑Z

Arrivé à ce stade, il est présumé que vous êtes capables de corriger vos erreurs en édition... Vous en ferez immanquablement car tout le monde en fait (voir la preuve ci-dessus !).

Sans trop entrer dans les détails, voici quelques mots sur la structure du programme (mais vous avez aussi le droit de lire les commentaires).

- les adresses 100 à 107 appellent la routine BDOS d'entrée d'une ligne au clavier (voir appendices). Le double registre DE doit pointer sur un bloc de commande contenant la longueur maximum de la ligne (50 caractères dans notre cas),

- la ligne située à l'adresse 108 émet les codes CR (retour chariot) et LF (nouvelle ligne) vers l'écran,

- les trois instructions suivantes testent ensuite si le premier caractère entré est un "#",

- après cela, il faut placer le symbole "#" à la fin de la ligne de texte (c'est une exigence de la routine BDOS d'affichage),

- le texte est alors envoyé sur l'écran (adresses 11F à 126), suivi des codes CR/LF (non émis par la routine),

- le compteur de caractères est ensuite remis à zéro et le programme bouclé ainsi jusqu'à entrée d'un caractère "#".

Nous allons maintenant *assembler ce programme*, en spécifiant à l'assembleur qu'il doit générer le fichier objet (.HEX) sur la disquette A:, et le fichier liste sur l'écran de la console.

```
A)asm prog.aax
CP/M ASSEMBLER - VER 2.0

 0100                    org     100h     ; debut TPA
                 ;
 0000 =          exit    equ     0        ; boot a chaud
 0005 =          bdos    equ     5        ; point entree BDOS
 000A =          input   equ     10       ; routine entree ligne
 0009 =          print   equ     9        ; routine sortie ligne
                 ;
 0100 113901     prog    lxi     d,bufrec         ; bloc commande entree
 0103 0E0A               mvi     c,input          ; code entree
 0105 CD0500             call    bdos             ; appel bdos
                 ;
 0108 CD3101             call    ligne            ; envoi cr/lf
                 ;
 010B 3A3B01             lda     texte            ; premier caractere texte
 010E FE23               cpi     '#'              ; est-ce # ?
 0110 CA0000             jz      exit             ; oui
                 ;
 0113 213B01             lxi     h,texte          ; debut texte
 0116 3A3A01             lda     ncr              ; nbre carac recus
 0119 5F                 mov     e,a              ; dans registre E
 011A 1600               mvi     d,0              ; reg D =0
 011C 19                 dad     d                ; (HL)+(DE)--->HL
 011D 3624               mvi     m,'$'            ; fin texte par $
                 ;
 011F 113B01             lxi     d,texte          ; pointe debut texte
 0122 0E09               mvi     c,print          ; code sortie
 0124 CD0500             call    bdos             ; appel bdos
                 ;
 0127 CD3101             call    ligne            ; envoi cr/lf
                 ;
 012A AF                 xra     a                ; 0--> nbre carac recus
 012B 323A01             sta     ncr              ;
 012E C30001             jmp     prog             ; suite
                 ;
 0131 116E01     ligne   lxi     d,crlf           ; routine envoi cr/lf
 0134 0E09               mvi     c,print          ; code sortie
 0136 C30500             jmp     bdos             ; appel bdos + retour
```

```
                ;
                ; zones des donnees ---------------
                ;
 0139 32        bufrec  db      50      ; longueur buffer
 013A 00        ncr     db      0       ; nbre carac recus
 013B           texte   ds      51      ; buffer texte
 016E 0D0A24    crlf    db      0dh,0ah,'$'
 0171                   end     prog
0171
003H USE FACTOR
END OF ASSEMBLY

A>
```

Le programme assembleur est capable de détecter quelques erreurs dont la liste est donnée en appendices.

Remarquez, à la fin du listing, la valeur 171 (hexa) qui indique l'adresse du premier emplacement mémoire libre après le programme, et l'énigmatique "003H USE FACTOR" qui correspond à un facteur de remplissage d'une table de l'assembleur.

Mais voyons un peu le catalogue :

```
A>dir prog.*
A: PROG      BAK : PROG      ASM : PROG      HEX
A>
```

Avez-vous noté l'existence du fichier PROG.BAK ? L'éditeur a bien fait son travail !

Comme prévu, l'assembleur a créé un fichier objet sous le nom : PROG.HEX. Mais à quoi peut-il bien ressembler, ce fichier ? A ceci :

```
A>type prog.hex
:100100001139010E0ACD0500CD31013A3B01FE2324
:10011000CA0000213B013A3A015F1600193624114A
:100120003B010E09CD0500CD3101AF323A01C300CC
:0B013000001116E010E09C30500320032
:03016E000D0A2453
:00010000FF

A>
```

Notez que nous venons d'afficher sur l'écran - et sans le moindre problème - un programme objet dont on pourrait s'attendre à ce que le format soit du pur binaire !

En réalité, ce format est connu sous le nom de "*format hexadécimal Intel*", du nom du constructeur de l'ordinateur MDS-800 sur lequel CP/M a vu le jour.

Selon ce format, chaque quartet (4 bits) binaire est codé en hexadécimal sur un octet ASCII ("0" à "F").

Par exemple :

le quartet : 1 0 1 1 (OBH en hexa) est codé

sur un octet : 0 1 0 0 0 0 1 0, ce qui

correspond au "B" en code ASCII

Le "**dump**" du disque, (fonction **DUMP** du CP/M) adopte ce même principe pour visualiser des valeurs binaires.

L'intérêt de ce format est de rendre les programmes ainsi codés, manipulables par des organes ne "digérant" habituellement que des codes ASCII (écran, Télétype, perforateur/lecteur de ruban, etc...).

Toujours selon ce format, le programme objet est découpé en blocs de 16 octets utiles, habillés d'une "enveloppe de protection" (une ligne sur le listing ci-dessus). Chaque bloc comprend l'adresse d'implantation en mémoire et un octet de checksum (somme de contrôle) qui permet de vérifier la validité du bloc. Voir appendices pour plus de détails.

Notre programme n'est toujours pas exécutable, mais il n'a jamais été aussi près de l'être !

EN RESUME

ASM - programme assembleur	
SYNTAXE	EXEMPLES
ASM <reference explicite>.<SOL> S = loc. source O = dest. objet L = dest. liste	ASM PROG ASM PROG.ABX ASM PROG.AAZ ASM B:PROG.BBA
commande transitoire	

LOAD - commande de conversion de fichiers

Pour *rendre notre programme exécutable* - et créer du même coup une nouvelle commande transitoire de CP/M consistant à recopier sur l'écran une ligne entrée au clavier (... bof !) - il suffit d'*utiliser la commande* **LOAD** de la manière suivante, et sans spécifier le type de fichier qui devra automatiquement être HEX sur la disquette :

```
A>load prog

FIRST ADDRESS 0100
LAST  ADDRESS 0170
BYTES READ    003E
RECORDS WRITTEN 01


A>
```

Cette commande, à partir du fichier PROG.HEX, va produire sur la même disquette que ce dernier, un fichier PROG.COM, comme le montre la scène suivante intitulée "la famille s'agrandit..." :

```
A>dir prog.*
A: PROG     BAK : PROG     ASM : PROG     HEX : PROG     COM
A>
```

Executable

Cette fois, il s'agit d'un vrai fichier binaire (essayez **TYPE...**) que l'on peut toutefois inspecter par la commande **DUMP** :

```
A>dump prog.com

0000 11 39 01 0E 0A CD 05 00 CD 31 01 3A 3B 01 FE 23
0010 CA 00 00 21 3B 01 3A 3A 01 5F 16 00 19 36 24 11
0020 3B 01 0E 09 CD 05 00 CD 31 01 AF 32 3A 01 C3 00
0030 01 11 6E 01 0E 09 C3 05 00 32 00 00 39 00 39 00
0040 39 00 39 00 39 00 39 00 39 00 39 00 39 00 39 00
0050 39 00 39 00 39 00 39 00 39 00 39 00 39 00 39 00
0060 39 00 39 00 39 00 39 00 39 00 39 00 39 00 0D 0A
0070 24 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

A>
```

dernier octet du programme

Amusez-vous à en comparer le contenu avec le code généré du listing d'assemblage. Mais n'attendons pas plus longtemps pour essayer notre nouvelle commande CP/M...

```
A>prog
```

Etant donné qu'il ne se passe rien, ce qui à-priori semble normal, frappons un message ou deux, terminé par la touche Return. En plaçant le signe "#" au début de la phrase, le contrôle est rendu à CP/M.

```
Cette phrase sera repetee par le programme PROG...
Cette phrase sera repetee par le programme PROG...
How do you do ?
How do you do ?
#

A>
```

C'est fini !

EN RESUME

LOAD - commande de conversion de fichiers HEX --> COM	
SYNTAXE	**EXEMPLES**
LOAD <ref explicite sans le type>	LOAD PROG LOAD B:PROG
commande transitoire	

DDT - utilitaire de mise au point

Notre programme fonctionne correctement, ce qui n'est pas tout à fait étonnant étant donnée sa taille. Le cas échéant, CP/M nous fournit *un puissant outil de mise au point* (debug = debogage, retirer les bogues (les erreurs)) qui nous donne aussi une idée plus précise de ce qui peut se passer réellement au niveau du microprocesseur, durant l'exécution d'un programme. Il s'agit de DDT (Dynamic Debugging Tool = outil de débogage dynamique).

Le chargement de **DDT** provoque un "écrasement" de la zone **CCP**, afin de laisser **TPA** libre pour recevoir le programme à tester.

L'appel se fait ainsi :

DDT <nom du programme à tester>

Essayons cela sur notre programme :

```
A>DDT PROG.COM
DDT VERS 2.2
NEXT  PC
0180 0100
-
```

DDT affiche deux valeurs : la prochaine adresse libre après le programme chargé (NEXT) et l'adresse de début d'exécution chargée dans le compteur programme (PC).

Ce programme de mise au point est équipé d'un certain nombre de commandes dont nous allons examiner les plus simples et les plus usuelles (voir chapitre 4 pour les commandes complémentaires).

Le signe "-" indique que DDT attend un ordre. Commençons donc par *la commande* **D** (Dump) qui va nous *fournir un dump hexadécimal de la mémoire :*

```
-D
0100 11 39 01 0E 0A CD 05 00 CD 31 01 3A 3B 01 FE 23 .9.......1.:;..#
0110 CA 00 00 21 3B 01 3A 3A 01 5F 16 00 19 36 24 11 ...!;.::._....6$.
0120 3B 01 0E 09 CD 05 00 CD 31 01 AF 32 3A 01 C3 00 ;.......1..2:...
0130 01 11 6E 01 0E 09 C3 05 00 32 00 00 39 00 39 00 ..n......2..9.9.
0140 39 00 39 00 39 00 39 00 39 00 39 00 39 00 39 00 9.9.9.9.9.9.9.9.
0150 39 00 39 00 39 00 39 00 39 00 39 00 39 00 39 00 9.9.9.9.9.9.9.9.
0160 39 00 39 00 39 00 39 00 39 00 39 00 39 00 0D 0A 9.9.9.9.9.9.9...
0170 24 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 $...............
0180 1A 84 12 13 C3 69 01 D1 2E 00 E9 0E 10 CD 05 00 .....i..........
0190 32 5F 1E C9 21 66 1E 70 2B 71 2A 65 1E EB 0E 11 2_..!f.p+q*e....
01A0 CD 05 00 32 5F 1E C9 11 00 00 0E 12 CD 05 00 32 ...2_...........2
0180 5F 1E C9 21 68 1E 70 2B 71 2A 67 1E EB 0E 13 CD _..!h.p+q*g.....
-
```

Lorsque cette commande n'est accompagnée d'aucun paramètre, elle affiche douze lignes à partir de l'adresse 100H (16 dit la documentation Digital Research ?). Remarquez l'interprétation ASCII dans la zone droite du dump. Elle ne nous est pas très utile pour l'instant. Les valeurs qui suivent le programme (à partir de l'adresse 180) sont sans signification.

Mais il y a mieux ! *La commande* **L** nous fournit *une liste désassemblée de notre programme* à partir de l'adresse donnée par PC :

```
-L
 0100  LXI  D,0139
 0103  MVI  C,0A
 0105  CALL 0005
 0108  CALL 0131
 010B  LDA  013B
 010E  CPI  23
 0110  JZ   0000
 0113  LXI  H,013B
 0116  LDA  013A
 0119  MOV  E,A
 011A  MVI  D,00
-
```

Une nouvelle commande L nous donnerait les instructions suivantes. Il est à noter que les étiquettes (labels) ne sont pas présents dans cette liste, mais nous reconnaissons bien là, notre programme d'origine, sous une forme, certes, un peu plus dépouillée.

Il est possible, avec DDT, de *modifier les données du programme examiné*. Cela se fait simplement par *la commande* **S** (substitution) sous cette forme :

S<adresse à examiner> <retour chariot>

DDT affiche alors le contenu de l'adresse spécifiée et attend l'une des trois actions suivantes :

<retour chariot> : passage à l'adresse suivante
<nouvelle valeur> <retour chariot> : substitution
<caractère "."> : on quitte la commande S.

Nous allons, par exemple, remplacer le caractère "#" (code hexa 23H) par le caractère "/" (code hexa 2FH), et *lancer notre programme* ainsi modifié, par une nouvelle commande : **G** (Go = branchement à l'adresse spécifiée) :

```
S10E
010E FE
010F 23 2F
0110 CA ^Z
?
-G100
QWERTYUIOP
QWERTYUIOP
#
#
/
A>
```

'/'

il vaut mieux utiliser 4 "."...

ça marche !

Comme il est constaté ci-dessus, la frappe d'un caractère "étranger" provoque la sortie de la commande S, accompagné d'un signe d'incompréhension de DDT.

Après l'ordre de branchement, le contrôle est donné au programme PROG et nous constatons cette fois-ci que le code "#" n'est plus interprété comme une condition de sortie, alors que "/" l'est bien. Notre modification a fonctionné.

Le contrôle étant rendu à CP/M, nous allons rappeler DDT en utilisant une manière différente de la première :

```
A>DDT
DDT VERS 2.2
-
```

Tout se passe comme plus haut mais aucun programme n'est chargé en mémoire.

Pour *appeler un programme, il faut en définir le nom par la commande* **I** (input = entrée), *puis provoquer la lecture disquette par la commande* **R** (Read = lecture) :

```
 IPROG.COM
-R
NEXT  PC
0180 0100
-
```

Nous nous retrouvons alors dans les mêmes conditions que précédemment :

```
L100
 0100  LXI  D,0139
 0103  MVI  C,0A
 0105  CALL 0005
 0108  CALL 0131
 010B  LDA  013B
 010E  CPI  23
 0110  JZ   0000
 0113  LXI  H,013B
 0116  LDA  013A
 0119  MOV  E,A
 011A  MVI  D,00
-
```

La nouvelle commande ***X*** *va nous permettre d'examiner les registres* du microprocesseur :

```
 X
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI  D,0139
-
```

"COZOMOEOIO" signifie :

indicateur C à O
indicateur Z à O
indicateur M à O
indicateur E à O
indicateur I à O

B correspond aux registres BC, D aux registres DE, H aux registres HL. S est l'adresse de la pile (Stack) et P le compteur programme. L'instruction pointée par PC est également donnée sous forme désassemblée.

Se reporter aux ouvrages traitant des microprocesseurs 8080, 8085 ou Z80 pour de plus amples informations.

Mais la commande la plus intéressante de DDT est *la commande* **T** (trace) *qui permet de tracer le déroulement d'un programme,* instruction par instruction et de visualiser du même coup, l'évolution des registres :

```
 T
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI  D,0139*0103
-T
COZOMOEOIO A=00 B=0000 D=0139 H=0000 S=0100 P=0103 MVI  C,0A*0105
-
```

Nous venons ci-dessus, d'exécuter les deux premières instructions du programme PROG...

Noter l'évolution des registres D et E, du compteur programme et des instructions désassemblées. La dernière valeur (précédée du signe "*"), indique l'adresse de la prochaine instruction à exécuter.

Pour lancer l'exécution des 16 instructions suivantes (10 en hexa), il suffit de frapper **T10** :

```
 T10
COZOMOEOIO A=00 B=000A D=0139 H=0000 S=0100 P=0105 CALL 0005
COZOMOEOIO A=00 B=000A D=0139 H=0000 S=00FE P=0005 JMP  2C00
COZOMOEOIO A=00 B=000A D=0139 H=0000 S=00FE P=2C00 JMP  32A2
COZOMOEOIO A=00 B=000A D=0139 H=0000 S=00FE P=32A2 XTHL
COZOMOEOIO A=00 B=000A D=0139 H=0108 S=00FE P=32A3 SHLD 3B4A
COZOMOEOIO A=00 B=000A D=0139 H=0108 S=00FE P=32A6 XTHL
COZOMOEOIO A=00 B=000A D=0139 H=0000 S=00FE P=32A7 JMP  3C06
COZ1MOE1IO A=00 B=000D D=0139 H=0000 S=0100 P=0108 CALL 0131
COZ1MOE1IO A=00 B=000D D=0139 H=0000 S=00FE P=0131 LXI  D,016E
COZ1MOE1IO A=00 B=000D D=016E H=0000 S=00FE P=0134 MVI  C,09
COZ1MOE1IO A=00 B=0009 D=016E H=0000 S=00FE P=0136 JMP  0005
COZ1MOE1IO A=00 B=0009 D=016E H=0000 S=00FE P=0005 JMP  2C00
COZ1MOE1IO A=00 B=0009 D=016E H=0000 S=00FE P=2C00 JMP  32A2
COZ1MOE1IO A=00 B=0009 D=016E H=0000 S=00FE P=32A2 XTHL
COZ1MOE1IO A=00 B=0009 D=016E H=010B S=00FE P=32A3 SHLD 3B4A
COZ1MOE1IO A=00 B=0009 D=016E H=010B S=00FE P=32A6 XTHL *32A7
-
```

Notez que la prochaine adresse de l'instruction à exécuter ne figure que sur la dernière ligne.

La commande **G** vue plus haut, permet aussi de définir un point d'arrêt (breakpoint) lorsque le programme arrive à une adresse donnée. Attention : si le programme ne passe jamais à cette adresse, le contrôle DDT sera perdu...

Lançons maintenant l'exécution à partir du début, en spécifiant un point d'arrêt à l'adresse 11F, juste avant l'affichage du texte (voir listing d'assemblage plus haut), et frappons une phrase :

```
 G100,11F
ESPRIT ES-TU LA ?
*011F
-
```

L'arrêt est signalé par un "*" suivi de l'adresse correspondante, et le contrôle est donné de nouveau à DDT. A ce stade, nous pouvons effectuer quelques vérifications, comme par exemple un dump entre les adresses 139H et 171H, au moyen de la commande **D** :

il est là !

'$'

```
 D139,171
0139 32 11 45 53 50 52 49 2.ESPRI
0140 54 20 45 53 2D 54 55 20 4C 41 20 3F 24 00 39 00 T ES-TU LA ?$.9.
0150 39 00 39 00 39 00 39 00 39 00 39 00 39 00 39 00 9.9.9.9.9.9.9.9.
0160 39 00 39 00 39 00 39 00 39 00 39 00 39 00 0D 0A 9.9.9.9.9.9.9...
0170 24 00 $.
-
```

La portion interprétée en ASCII nous montre que notre texte figure dans la zone buffer, et qu'il est terminé par le caractère "**$**".

Pour continuer l'exécution, frappons **G** sans paramètre (plus de condition d'arrêt).

```
 G
ESPRIT ES-TU LA ?
#

A>
```

Vous pouvez vous amuser très longtemps avec ces quelques commandes de DDT... N'hésitez pas !

EN RESUME

DDT - utilitaire de mise au point	
SYNTAXE	**EXEMPLES**
DDT DDT <ref explicite>	DDT DDT PROG.COM DDT B:PROG.COM
commande transitoire	

Commandes usuelles DDT	Fonctions realisees
D D<ad debut> D<ad debut>,<ad fin>	- Dump memoire en hexa et en ASCII
L L<ad debut> L<ad debut>,<ad fin>	- Liste symbolique (desassemblage)
S<ad debut>	- Substitution memoire (arret par ".")
G G<ad debut> G<ad debut>,<ad arret>	- Branchement a une adresse donnee (ou PC) avec point d'arret
I<ref explicite>	- Definition d'un nom de fichier
R	- lecture fichier defini par R
X	- liste contenu registres du microprocesseur
T<nombre de lignes>	- Trace instruction par instruction

SAVE - commande de transfert de mémoire à disque

Il est intéressant de pouvoir effectuer des modifications dans un programme, sans repasser par les phases ED + ASM + LOAD. Il serait plus intéressant encore de pouvoir sauvegarder ces modifications (souvenez-vous du "#" remplacé par le "/" par exemple).

La commande **SAVE** que nous allons aborder maintenant, nous permet d'exaucer ce souhait.

Revenons un peu en arrière, et apportons de nouveau cette petite modification :

```
A>ddt prog.com
DDT VERS 2.2
NEXT  PC
0180 0100
-s10f
010F 23 2f
0110 CA .
-g0
```

pour terminer

Le "**g0**" (branchement en zéro) nous permet de sortir de DDT en exécutant un démarrage à chaud.

Pour sauvegarder notre programme ainsi modifié (donnons-lui le nom PROG1 par exemple), il suffira de frapper la commande suivante :

```
A>save 1 prog1.com
A>
```

Le "1" indique au programme **SAVE** qu'il faut sauver 1 "page" de la zone TPA (commençant en 100H) sur disque. On entend par "**page**", *une zone mémoire de 256 octets*. Toute sauvegarde réalisée par la commande **SAVE** sera toujours un multiple de 256 octets.

Notre programme ayant une longueur de 80H (soit 128 octets), une page est largement suffisante pour le contenir.

Nous pouvons maintenant l'exécuter et vérifier que cette modification fonctionne correctement :

```
prog1
asdfghjkl
asdfghjkl
#
#
/

A>
```

C'est bon !

Revenons en arrière, au début de ce chapitre, plus exactement à *la commande* **USER**. Avec cette commande, nous savions passer dans une autre zone utilisateur, et lister le catalogue qui reste désespérément vide.

Nous allons voir maintenant, comment la commande SAVE nous permet de faire entrer le "loup dans la bergerie" !

Chargeons tout d'abord **PIP** en mémoire, sans l'exécuter. Comment ? Par **DDT** bien sûr ! :

```
A>ddt pip.com
DDT VERS 2.2
NEXT  PC
1E00 0100
-g0
A>
```

Déplaçons-nous ensuite en zone 1 et vérifions que le catalogue est toujours vide :

```
A>user 1
A>dir
NO FILE
A>
```

La commande **SAVE** (qui fort heureusement est résidente...) va nous permettre de sauver **PIP** en zone 1 :

```
A>save 29 pip.com
A>dir
A: PIP      COM
A>
```

= 1EH - 1 en decimal

La valeur du nombre de pages est calculée en prenant la valeur donnée par NEXT dans DDT et en lui retirant 100H (on obtient 1D00H). L'octet de poids fort de cette valeur (1DH) converti en décimal donne 29 (16 et 13).

La commande **DIR** nous confirme alors que le catalogue n'est plus vide maintenant, puisqu'il contient **PIP.** Nous verrons plus loin comment ce programme nous permettra de transférer d'autres programmes en zone 1.

La commande **STAT USR:** (transitoire, ne figurant pas en zone 1) nous indique que deux zones sont définies maintenant :

```
A>user 0
A>stat usr:

Active User : 0
Active Files: 0 1
A>
```

Voici un petit truc permettant de relancer un programme resté en zone TPA. Il suffit de sauver un nombre de page nul... comme ceci :

```
A>save 0 encore.com
A>
```

Lorsque nous appellerons ce programme (cette nouvelle commande, dirons-nous !), CP/M ne charge rien en zone TPA... et donne le contrôle à l'instruction située à l'adresse 100H (le programme précédemment chargé). Essayons cela :

```
A>prog
je reste calme !
je reste calme !
#

A>encore
je reste tres calme...
je reste tres calme...
#
A>
```

Voici maintenant la méthode à employer pour *copier des programmes en zone 1* (ou toute autre zone qui contient déjà **PIP**).

Le programme **PIP** possède un paramètre (G) qui, comme tout paramètre (voir chapitre 4) doit être formulé entre crochets. Ce paramètre permet de spécifier la zone source dans laquelle le programme devra être cherché. Dans notre cas, ce sera la zone 0.

Essayons de transférer en zone 1 notre programme PROG et exécutons-le :

```
A>user 1
A>pip prog.com=prog.com[GO]

A>dir
A: PIP      COM : PROG     COM
A>prog
prog fonctionne en zone 1...
prog fonctionne en zone 1...
#

A>user 0
A>
```

Attention : la zone paramètre entre crochets doit être collée au nom du fichier (pas d'espace).

EN RESUME

```
+-------------------------------------------------------+
!  SAVE - commande de transfert memoire --> disque      !
+-------------------------+-------------------------+
!         SYNTAXE         !        EXEMPLES         !
+-------------------------+-------------------------+
! SAVE <n> <ref explicite> !  SAVE 1 PROG.COM        !
!                         !  SAVE 29 PIP.COM        !
!   n=nbre de pages       !                         !
!     de 256 octets       !                         !
+-------------------------+-------------------------+
!  commande residente                               !
+-------------------------------------------------------+
```

XSUB - extension de la commande SUBMIT (V2)

Rappelez-vous la commande **SUBMIT.** Elle permet d'enchainer des évènements prédéterminés, contenus dans un fichier de commandes.

Pourquoi ne pas exécuter le programme PROG en plaçant dans le fichier de commandes une phrase devant habituellement être frappée au clavier et en terminant le tout par le signe "**#**" pour quitter le programme ?

Créons d'abord le fichier de commande :

```
A>ed essai.sub

NEW FILE
     : *i
    1:  stat
    2:  prog
    3:  $1
    4:  #
    5:  stat
    6:  ↑Z
     : *e

A>
```

... et lançons-le (la phrase devant être affichée sera : "cp/m") :

```
A>submit essai cp/m

A>STAT
A: R/W, Space: 140k

A>PROG
frappe au clavier...
frappe au clavier...
#

A>CP/M
CP/M?

A>
```

entrée clavier

c'est le $1

là, il ne comprend pas !

Oh ! Mais que se passe t-il ? Cela ne semble pas fonctionner comme prévu, et il est toujours nécessaire d'entrer la phrase au clavier, comme auparavant, cette dernière n'ayant pas été prélevée dans le fichier de commandes. Après la frappe du symbole "#", l'enchainement automatique se poursuit et CP/M doit interpréter la "commande"... "cp/m" qui doit le plonger dans une profonde stupéfaction ! Le fichier de commandes est alors abandonné, et le contrôle est rendu à CP/M.

En conclusion, l'appel de la routine BDOS (code 10) d'entrée de caractères n'est pas interceptée par **SUBMIT.**

Il y a heureusement une solution depuis la version 2 du CP/M, c'est **XSUB.** *Lorsque* **XSUB** *est placé en tête du fichier de commandes, la routine BDOS d'entrée de caractères est interceptée par SUBMIT et les données fournies dans le fichier de commandes sont prises comme si elles venaient du clavier.*

Démonstration :

```
A>ed essai.sub
     : *i
    1:  xsub
    2:
     : *#a
    2: *e

A>
```

on ajoute XSUB au début du fichier

↑Z

Nous insistons : **XSUB** doit être placé en tête du fichier de commandes.

Essayons de nouveau...

```
A>submit essai cp/m

A>XSUB

A>STAT
A: R/W, Space: 139k


A>PROG
CP/M

CP/M
#


(xsub active)
A>STAT
A: R/W, Space: 139k


A>
```

Cette fois, c'est bon !

warm boot

Tout se déroule maintenant comme prévu ! Notez que le message "xsub active" apparait à chaque fois qu'un démarrage à chaud se produit pendant l'exécution du fichier de commandes.

EN RESUME

XSUB - extension de la commande SUBMIT (V2)	
SYNTAXE	EXEMPLES
XSUB (doit etre place au DEBUT du fichier de commandes)	XSUB ... fichier de commandes
commande transitoire (Version 2 uniquement)	

MOVCPM - commande de configuration CP/M

Les deux commandes qui suivent ne sont pas d'une utilisation courante : elles sont employées pour *configurer ou pour générer un nouveau système CP/M.*

A l'origine, la disquette CP/M fournie par Digital Research est configurée pour "tourner" sur une configuration disposant d'un minimum de mémoire. Le système CP/M étant placé en fond de mémoire, l'utilisateur disposant d'une capacité mémoire plus importante doit pouvoir déplacer le système là où il le souhaite.

Cette fonction s'effectue par la commande **MOVCPM**, qui contient en elle-même un système CP/M et une table des adresses à reloger (adresse devant être modifiées lors d'un déplacement).

Deux paramètres optionnels peuvent être donnés à **MOVCPM** par l'utilisateur : la configuration souhaitée du système (en K octets) et un indicateur spécifiant si le nouveau système doit être placé ou non en zone TPA pour sauvegarde ultérieure.

```
A>movcpm

CONSTRUCTING 62k CP/M vers 2.2

62k CP/M vers 2.2

A>
```

Sans paramètres, **MOVCPM** construit un système provisoire (non sauvegardable) logé en fond de mémoire. En général, les deux derniers K octets sont réservés pour la mémoire écran ou pour la ROM (moniteur dans le MDS 800). Le contrôle est alors donné au nouveau système.

```
A>movcpm 48

CONSTRUCTING 48k CP/M vers 2.2

48k CP/M vers 2.2

A>
```

Dans l'exemple ci-dessus, la configuration 48K est imposée à **MOVCPM.**

```
A>movcpm 48 *

CONSTRUCTING 48k CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM48.COM"
A>
```

Construction d'une configuration 48K, mais l'astérisque stipule que le système doit être sauvegardé. Il sera disponible en zone TPA pour être traité par **SAVE** (34 pages à sauvegarder) ou par **SYSGEN** que nous verrons plus loin. Le nouveau système n'est pas exécuté.

Attention : il ne faut rien charger en zone TPA avant la sauvegarde...

```
A>movcpm * *

CONSTRUCTING 62k CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM62.COM"
A>
```

Construction d'une configuration maximale avec sauvegarde ultérieure.

EN RESUME

MOVCPM - commande de configuration CP/M	
SYNTAXE	EXEMPLES
MOVCPM MOVCPM <par1> <par2> par1 : taille en K par2 : sauvegarde	MOVCPM MOVCPM 48 MOVCPM 48 * MOVCPM * *
commande transitoire	

SYSGEN - commande de génération de système

Pour copier une disquette ou construire un système de configuration donnée, il est nécessaire d'utiliser des disquettes formattées selon des normes données par le constructeur de l'ordinateur (IBM 3740 pour les disquettes 8 pouces).

En principe, celui-ci fournit un programme utilitaire souvent appelé **FORMAT** ou **INIT**, capable d'assumer cette fonction. Toutefois, cette commande n'est pas incluse au CP/M commercialisé par Digital Research et reste, de même que le BIOS, à la charge du constructeur de l'ordinateur employé.

La nouvelle configuration étant construite, il peut être utile de la conserver de manière permanente sur une disquette de travail.

Nous pouvons réaliser cela par la commande **SYSGEN** qui peut également être utilisée pour effectuer une simple copie :

```
A>sysgen
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)a
SOURCE ON A, THEN TYPE RETURN
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)b
DESTINATION ON B, THEN TYPE RETURN
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

A>
```

Cette fonction peut être décomposée ainsi :

- lecture du système sur la disquette A: et transfert en zone TPA.
- écriture du système logé en zone TPA sur la disquette B:

La même opération peut être réalisée sur la même unité de disquette, en substituant les disquettes avant de répondre à la seconde question posée par **SYSGEN** :

source drive name : A
type RETURN

(remplacer la disquette)

destination drive name : A
type RETURN

Pour générer une disquette dans la configuration souhaitée, il faudra faire :

```
A>movcpm * *

CONSTRUCTING 62k CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM62.COM"
A>sysgen
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)b
DESTINATION ON B, THEN TYPE RETURN
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

A>
```

Return

Le système ainsi copié est placé sur les deux premières pistes de la disquette et contient également le "bootstrap" (programme de lancement du système à la mise sous tension).

Pour copier intégralement une disquette, la commande **SYSGEN** ne suffit pas car seul, le système est copié.

Il est donc nécessaire de compléter par :

```
A>pip b:=*.*

COPYING -
MOVCPM.COM
PIP.COM
SUBMIT.COM
XSUB.COM
ED.COM
ASM.COM
DDT.COM
LOAD.COM
STAT.COM
SYSGEN.COM
DUMP.COM
DUMP.ASM
BIOS.ASM
CBIOS.ASM
TOTO.ASC
PROG.BAK
DUMP.PRN
PROG.ASM
PROG.HEX
PROG.COM
PROG1.COM
ENCORE.COM
ESSAI.SUB
ESSAI.BAK

A>
```

Prenons maintenant notre nouvelle disquette, rechargeons entièrement le système, et consultons le catalogue :

```
62k CP/M vers 2.2

A>dir
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : TOTO     ASC : PROG     BAK
A: DUMP     PRN : PROG     ASM : PROG     HEX : PROG     COM
A: PROG1    COM : ENCORE   COM : ESSAI    SUB : ESSAI    BAK
A>


user 1
A>dir
NO FILE
A>
```

Remarque : les fichiers de la zone 1 n'ont pas été copiés.

EN RESUME

SYSGEN - commande de generation de systeme	
SYNTAXE	EXEMPLES
SYSGEN	SYSGEN
commande transitoire	

L'étude de ce chapitre s'achève ici. Vous devez en savoir suffisamment maintenant pour prétendre pratiquer le CP/M.

Il reste à examiner plus en détails certaines fonctionalités des programmes **ED**, **PIP** et **DDT** que nous allons étudier dans le chapitre suivant.

Mais avant d'aller plus loin, assurez-vous par un rapide coup d'oeil sur les tableaux "en résumé", que vous êtes bien à l'aise avec les différentes commandes que nous venons d'aborder. Ils sont là pour cela !

Si vous vous posez la moindre question, relisez les lignes qui précèdent le tableau et *reprenez les exemples sur votre système*.

...ENCORE UN PETIT EFFORT !

Dans le chapitre précédent, nous avons vu comment utiliser les programmes **ED**, **PIP** et **DDT**, tout au moins en ce qui concernait les utilisations les plus courantes et, avec ce que vous avez appris jusque là, vous pouvez déjà prétendre connaitre CP/M.

Nous vous proposons maintenant de continuer l'étude au second niveau de ces trois programmes.

Les choses que vous allez apprendre ne seront peut-être pas utilisées fréquemment, mais il est bon de savoir qu'elles existent le jour où on en a besoin. Dans tous les cas, cela vous permettra, par la même occasion, de pratiquer un peu plus longtemps, et - peut-être - de comprendre des choses que vous n'aviez pas comprises la première fois.

Un résumé des commandes CP/M figure en appendice : utilisez-le !

ED - second niveau

La figure 1 donne une petite idée sur les mécanismes de base de l'éditeur **ED.**

Le buffer mémoire, représenté au centre de la figure, est la zone mémoire dans laquelle se trouve chargé le fichier que vous êtes en train de créer ou de modifier. Cette zone, variable selon la taille mémoire de votre système, est forcément limitée. Cela signifie qu'elle ne sera pas toujours capable de contenir la totalité du fichier édité.

Pour les gros fichiers, la zone buffer est donc utilisée comme une "fenêtre" donnant accès à une portion seulement du fichier complet : le fichier original transite, morceau par morceau, dans cette zone - à la manière d'un film passant devant une fenêtre de lecture - et est écrit morceau par morceau, dans un fichier temporaire. Durant son passage dans la fenêtre, la portion du fichier peut être simplement visualisée ou recevoir des modifications. Mais, ce parcours ne s'effectue que dans un seul sens et on ne peut pas revenir en arrière, à moins de rebobiner tout le film...

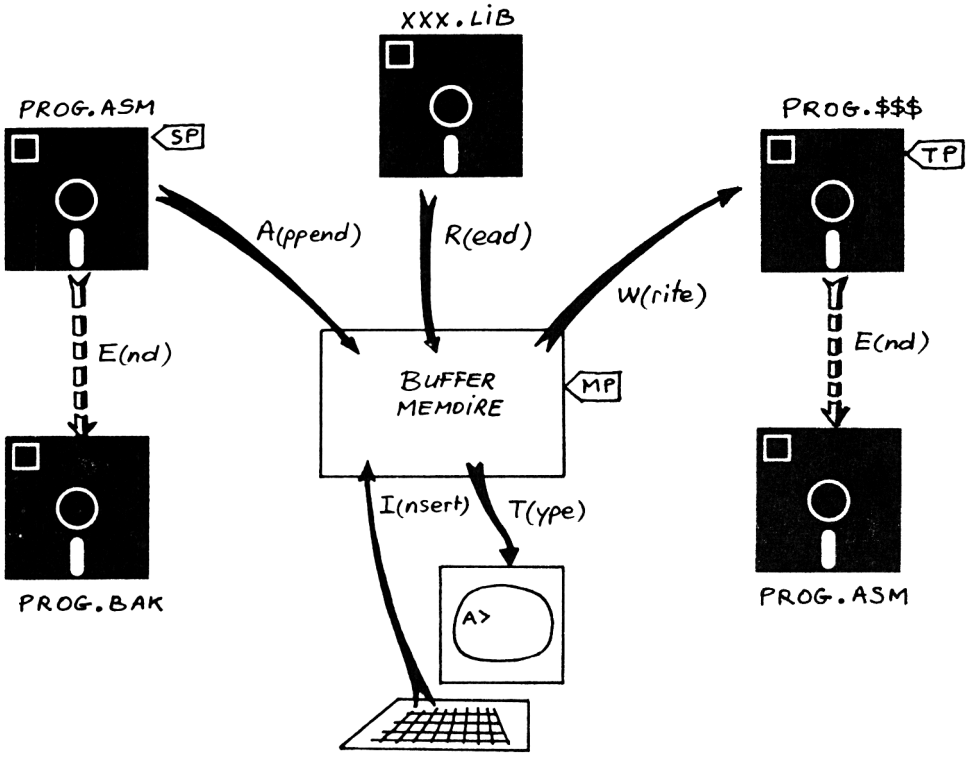


Figure 1

Sur notre figure, le programme PROG.ASM est en train d'être édité. Il passe par le buffer mémoire dans lequel il est appelé par la commande **Append** (**A**). Pendant cette période, la portion visualisée peut recevoir des modifications (insertion de textes par la commande **I**, insertion d'un fichier de la librairie par la commande **R**, suppression de lignes par la commande **K**), ou être affichée sur l'écran de la console (commande **T**).

Pour passer à la portion suivante, il est nécessaire de sauver le contenu du buffer mémoire dans un fichier temporaire, qui prendra le nom PROG.$$$. Cette action s'effectue par la commande **W** (Write). Bien entendu, si le buffer mémoire peut recevoir la totalité du fichier, cette opération n'est pas nécessaire.

A chaque "Append" dans le buffer mémoire, un pointeur est mis à jour dans le fichier source (SP) et se déplace vers la fin du fichier au fur et à mesure des transferts. Il en est de même pour le fichier temporaire PROG.$$$ et son pointeur (TP), qui évolue au fur et à mesure des "write".

Dans le cas où le fichier ne subit pas d'altérations, les deux pointeurs SP et TP sont "synchronisés".

Le remplissage du buffer mémoire est, lui aussi, géré par un pointeur (MP). A chaque "append", il se déplace vers le fond du buffer et à chaque "write", il est ré-initialisé et repasse au début. Cela signifie aussi qu'à chaque "write", le buffer mémoire est vidé de son contenu pour céder la place à l'"append" suivant.

Naturellement, en création de fichier, le transfert fichier source/buffer mémoire (append) n'existe pas.

En fin d'édition (commande **E**), les actions suivantes se succèdent :

- un "write" est forcé si le buffer n'est pas vide
- le fichier PROG.ASM devient PROG.BAK (sauvegarde automatique de l'original)
- le fichier PROG.$$$ devient PROG.ASM (nouvelle version)

La figure 2 donne un exemple de ce qui se passe à la suite d'un "append" de deux lignes :

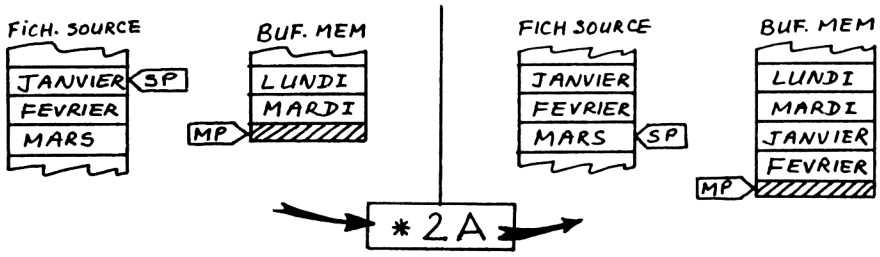


Figure 2

La partie gauche de la figure décrit une portion du fichier source contenant 3 lignes. Le pointeur source SP pointe sur la première de ces lignes (janvier). Le buffer mémoire, qui contient déjà 2 lignes, a son pointeur MP positionné sur une zone mémoire inutilisée. Après la commande "**2A**" (append de 2 lignes), la partie droite de la figure montre que SP et MP ont progressé de 2 lignes.

La figure 3 illustre le principe de la commande "write". Nous voyons à droite, que deux lignes sont transférées du buffer mémoire vers le fichier temporaire. Essayez de reproduire ce simple exemple sur votre système.

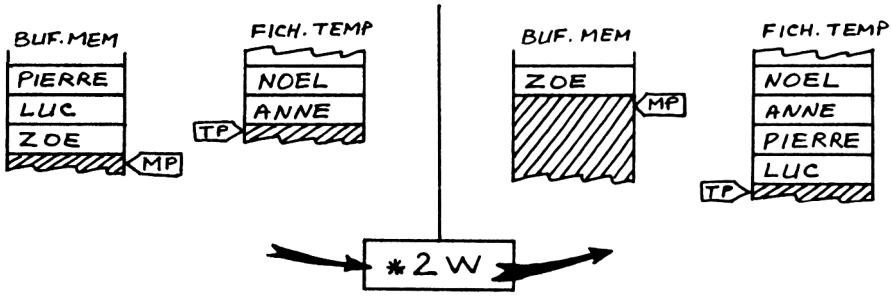


Figure 3

Les commandes **L** et **K** que nous avons déjà étudiées, et qui agissent sur les <u>lignes</u> du fichier, ont leur contrepartie : **C** et **D** effectuent respectivement un déplacement et une suppression au niveau des <u>caractères</u> d'une ligne. Comme le montre la figure 4, **C** permet d'avancer ou de reculer d'un nombre de caractères donné :

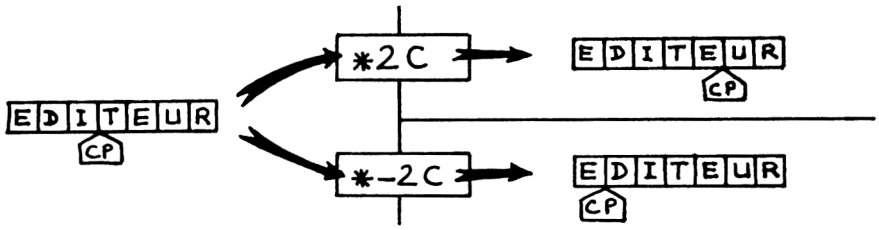


Figure 4

CP désigne cette fois, le **pointeur caractère** dans la ligne courante. La commande **T** ne sera effective qu'à partir de la position donnée par **CP**. Pour supprimer les caractères - situés avant ou après CP - on utilise la commande **D** (figure 5) :

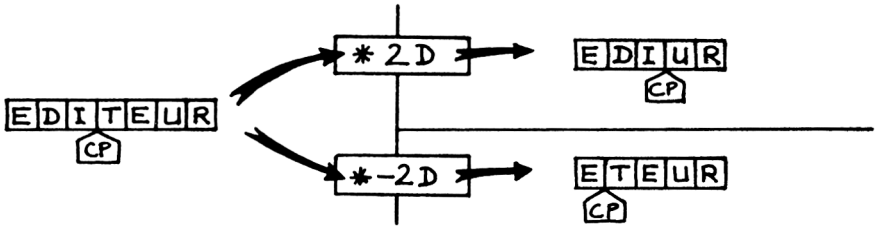


Figure 5

La commande **K**, comme nous l'avons vue, supprime des lignes du fichier. Mais qu'en est-il lorsque CP n'est plus situé au début de la ligne courante ? La réponse est donnée par la figure 6. Mais essayez donc sur votre système !

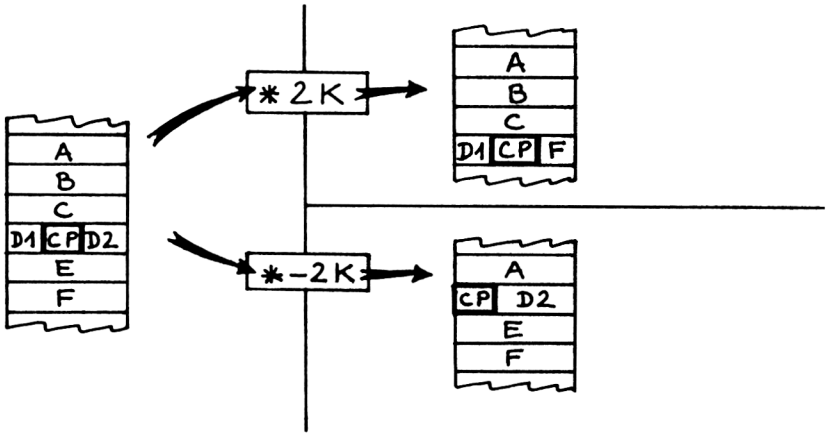


Figure 6

Remarquez que D1 et D2 représentent les champs situés avant et après le pointeur caractère CP, chaque champ pouvant contenir un ou plusieurs caractères.

Examinons ensuite le mécanisme de la commande **L** (figure 7) (déplacement du pointeur ligne courant) :

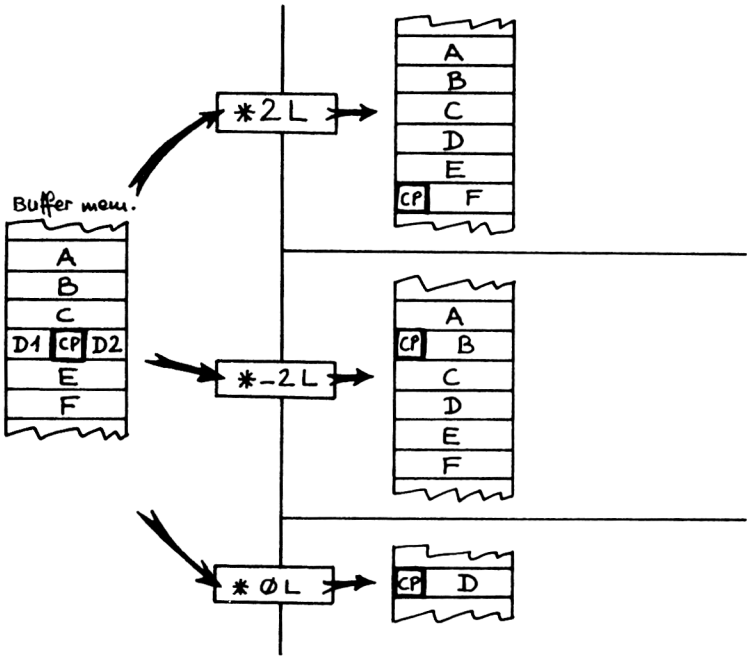


Figure 7

Nous voyons qu'après exécution, le pointeur CP est toujours situé en début de ligne. **OL** permet le retour en début de ligne courante. D1 et D2 représentent les portions gauche et droite du champ D.

Enfin, la commande **I** est représentée figure 8, au même titre que la commande **R** qui présente quelques similitudes avec elle :

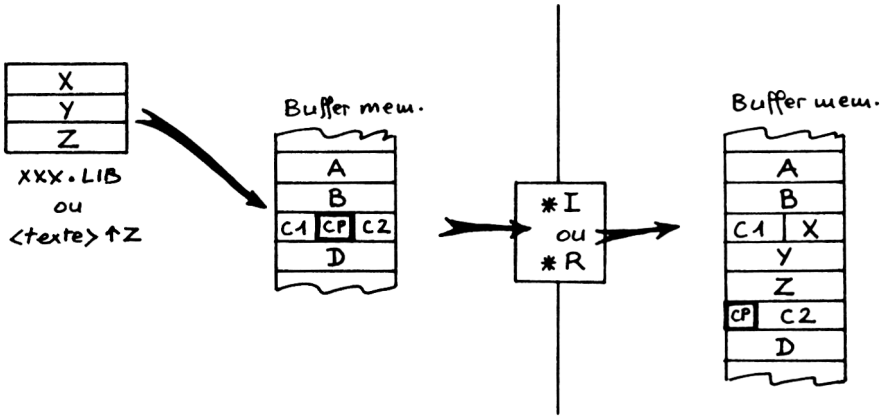


Figure 8

I permet d'insérer un texte (terminé par **CTRL Z**) à partir du pointeur CP, alors que **R** effectue le même genre d'insertion mais cette fois à partir d'un fichier se trouvant sur disquette et dont le type est ".LIB (library). Cela peut être pratique pour inclure une routine, par exemple, à un endroit donné du fichier.

Votre système, nous l'espérons, est toujours sous tension (?), alors commençons par créer deux petits fichiers qui vont nous servir par la suite :

```
A>ed mois.txt

NEW FILE
     : *i
    1:  janvier          fevrier          mars
    2:  avril            mai              juin
    3:  juillet          aout             septembre
    4:  octobre          novembre         decembre
    5:
     : *e      ↑Z

A>


A>ed jours.lib

NEW FILE
     : *i
    1:  lundi, mardi, mercredi
    2:  jeudi,vendredi,samedi
    3:  dimanche
    4:
     : *e      ↑Z

A>
```

Rappelons ensuite le fichier "mois.txt" par l'éditeur et expérimentons les commandes **A** et **L** :

```
A>ed mois.txt
     : *i
    1:  MOIS DE L'ANNEE ----------------------------------------
    2:
     : *2A     ↑Z
    2: *B#T
    1:  MOIS DE L'ANNEE ----------------------------------------
    2:  janvier          fevrier          mars
    3:  avril            mai              juin
    1: *3L
     : *I
    4:  FIN DU PREMIER SEMESTRE ----
    5:
     : *#AB#T  ↑Z
    1:  MOIS DE L'ANNEE ----------------------------------------
    2:  janvier          fevrier          mars
    3:  avril            mai              juin
    4:  FIN DU PREMIER SEMESTRE ----
    5:  juillet          aout             septembre
    6:  octobre          novembre         decembre
    1: *
```

Nous voyons au passage, qu'il est possible de cumuler plusieurs ordres dans la même ligne : **#AB#T** charge en mémoire toutes les lignes (**#**) du fichier source, le pointeur ligne est ensuite positionné au début du fichier, et toutes les lignes (**#**) sont affichées.

Ecrivons ensuite 4 lignes dans le fichier temporaire :

```
1: *4W
5: *B#T
5:  juillet          aout            septembre
6:  octobre          novembre        decembre
5: *
```

et affichons le contenu du buffer mémoire. Nous constatons que les 4 premières lignes ont disparu. Il ne sera plus question d'y avoir accès, sauf... si l'on utilise la commande **H** :

```
5: *H
 : *B#T
 : *#A#T
1:  MOIS DE L'ANNEE ----------------------------------------
2:  janvier          fevrier         mars
3:  avril            mai             juin
4:  FIN DU PREMIER SEMESTRE ----
5:  juillet          aout            septembre
6:  octobre          novembre        decembre
1: *
```

La commande **H** correspond à la commande **E** (fin d'édition), suivie d'un rappel de l'éditeur. Les modifications précédentes sont prises en compte, même si l'on quitte (**Q**) l'éditeur, mais le buffer est remis à zéro. Il est donc nécessaire d'invoquer la commande **A.**

```
1: *4:
4: *T
4:  FIN DU PREMIER SEMESTRE ----
4: *1:T
1:  MOIS DE L'ANNEE ----------------------------------------
1: *4::5T
4:  FIN DU PREMIER SEMESTRE ----
5:  juillet          aout            septembre
4: *
```

Pour positionner le pointeur ligne sur un numéro précis, il suffit, comme nous le voyons ci-dessus, de frapper ce numéro suivi du signe ":". Pour lister les lignes 4 à 5, il faudra employer la forme "4::5".

Insérons maintenant un texte au début de la ligne 4 :

```
4: *I
4:  C'EST LA
4: *OLT
4:  C'EST LA FIN DU PREMIER SEMESTRE ----
4: *
```

et listons celle-ci (notez le "OLT").

Vous décidez que toutes les modifications précédentes (depuis la dernière commande Append) n'ont pas lieu d'être et vous désirez revenir aux conditions initiales ? Très simple. Il suffit de frapper la commande "**0**". Ed étant prudent par principe, vous demande la confirmation... :

```
     4: *O
O-(Y/N)?Y
      : *B#T
      : *#A#T
     1:  MOIS DE L'ANNEE ----------------------------------------
     2:  janvier          fevrier          mars
     3:  avril            mai              juin
     4:  FIN DU PREMIER SEMESTRE ----
     5:  juillet          aout             septembre
     6:  octobre          novembre         decembre
     1: *
```

On recommence...

En répondant "Y" (yes), on repart au début.

Pour quitter l'éditeur (commande **Q**) sans tenir compte des éventuelles modifications apportées, c'est aussi simple :

```
     1: *Q
Q-(Y/N)?N
     1: *T
     1:  MOIS DE L'ANNEE ----------------------------------------
     1: *
```

on ne quitte pas encore...

mais cette fois, nous répondrons "N" (no).

Listons plutôt les deux lignes situées de part et d'autre de la ligne 4 :

```
1: *4:
4: *T
4:  FIN DU PREMIER SEMESTRE ----
4: *-2T
2:  janvier          fevrier          mars
3:  avril            mai              juin
4: *T
4:  FIN DU PREMIER SEMESTRE ----
4: *2T
4:  FIN DU PREMIER SEMESTRE ----
5:  juillet          aout             septembre
4: *
```

a moins que vous ne préfériez vous déplacer de deux lignes en avant ou en arrière :

```
4: *-2L
2: *T
2:  janvier          fevrier          mars
2: *2L
4: *T
4:  FIN DU PREMIER SEMESTRE ----
4: *
```

Nous pouvons aussi avancer CP de 10 caractères :

```
     4: *10C
     4: *T
MIER SEMESTRE ----
     4: *
```

Notez que "**T**" n'affiche que les caractères situés après CP. Très simple également, pour reculer de 2 caractères :

```
     4: *-2C
     4: *T
REMIER SEMESTRE ----
     4: *
```

avant d'en effacer 3... :

```
4: *3D
4: *OLT
4:  FIN DU PIER SEMESTRE ----
4: *
```

Le mot "premier" est maintenant estropié, mais nous allons réparer cela :

```
     4: *9C
     4: *T
ER SEMESTRE ----
     4: *I
     4:  REM
     4: *OLT
     4:  FIN DU PIREMER SEMESTRE ----
     4: *
```

C'est raté !

enfin presque ! (Authentique, ce n'était pas voulu ...). En fait, il n'y a pas d'espace au début de la phrase, comme nous l'avions pensé initialement. Mais cette situation ne va pas durer plus longtemps :

```
4: *8CD3CII^ZOLT
4:  FIN DU PREMIER SEMESTRE ----
4: *
```

Oufff !!

... Hein !!

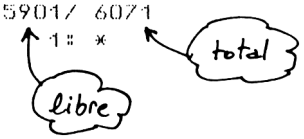
Un conseil : n'allez pas plus loin tant que vous n'avez pas compris la dernière commande !

L'éditeur ED est tout-à-fait prêt à supprimer les numéros de lignes dans le cas où ceux-ci vous gènent vraiment :

```
     4: *-V
*B#T
MOIS DE L'ANNEE ----------------------------------------
janvier          fevrier          mars
avril            mai              juin
FIN DU PREMIER SEMESTRE ----
juillet          aout             septembre
octobre          novembre         decembre
*V
     1: *
```

Mais c'est beaucoup moins pratique... Par contre, la forme "**OV**" est très utile :

```
   1: *OV
5901/ 6071
   1: *
```



puisqu'elle délivre le nombre d'octets restant libres, ainsi que la taille maximale du buffer mémoire (configuration 20K dans notre exemple).

Le moment est venu d'inclure le fichier "jours.lib" que nous avons créé plus haut, dans le buffer mémoire. Pour cela, positionnons CP à un endroit quelconque :

```
    1: *4:4CT
DU PREMIER SEMESTRE ----
    4: *OT
    4:  FIN *T
DU PREMIER SEMESTRE ----
    4: *
```

et insérons-le ainsi :

```
4: *RJOURS
7: *T
7:  DU PREMIER SEMESTRE ----
7: *B#T
1:  MOIS DE L'ANNEE ------------------------------------------------
2:  janvier          fevrier          mars
3:  avril            mai              juin
4:  FIN LUNDI, MARDI, MERCREDI
5:  JEUDI,VENDREDI,SAMEDI
6:  DIMANCHE
7:  DU PREMIER SEMESTRE ----
8:  juillet          aout             septembre
9:  octobre          novembre         decembre
1: *
```

Vous en conviendrez, l'endroit était assez mal choisi, car notre texte initial ne ressemble plus à rien ! Mais c'est le métier qui entre ! Supprimons donc l'intru :

```
1: *7:-2K
5: *4:4CK
4: *B#T
1:  MOIS DE L'ANNEE ------------------------------------------------
2:  janvier          fevrier          mars
3:  avril            mai              juin
4:  FIN DU PREMIER SEMESTRE ---
5:  juillet          aout             septembre
6:  octobre          novembre         decembre
1: *
```

Vous vous souvenez de la commande "**B**" ? Mais que peut donc faire la commande "**-B**" ?

```
1: *-B
 : *-LT
6:  octobre          novembre          decembre
6: *
```

Elle positionne le pointeur courant en fin de fichier, naturellement.

Après avoir appelé un fichier ".LIB" du disque, nous allons maintenant en sauver un :

```
6: *4:T
4:  FIN DU PREMIER SEMESTRE ----
4: *X
4: *
```

X$$$$$$$$.LIB

Dans le cas présent, nous ne sauvons qu'une seule ligne. Pour en sauver dix, il aurait fallu faire "**10X**". On ne peut pas choisir le nom du fichier : il s'appelle automatiquement "X$$$$$$$$.LIB". Cela nous permet ainsi de rappeler la ligne là où nous le désirons, en fin de buffer par exemple :

```
4: *-BR
 : *-2L2T
6:  octobre          novembre          decembre
7:  FIN DU PREMIER SEMESTRE ----
6: *
```

X$$$$$$$$.LIB

Mais cette fois, le mot "premier" n'est pas très approprié.

Essayons de le remplacer par le mot "second" :

```
6: *B
1: *FPREMIER^ZSPREMIER^ZSECOND
7: *0LT
7:  FIN DU SECOND SEMESTRE ----
7: *
```

Le pointeur se trouvant au début du fichier (B), il faut sauter le premier mot "premier" (c'est clair non ?). Pour cela, on utilise la commande de recherche (**F**). Ensuite, il est possible de substituer (**S**) "premier" par "second". Pour exécuter plusieurs fois cette opération, il suffira de placer le nombre correspondant devant la commande **S**.

Il faut insister sur un point . *la commande de recherche* **F** *ne s'effectue qu'à l'intérieur du buffer mémoire.* Lorsque le fichier est très gros et ne peut donc pas être contenu entièrement dans le buffer, on peut utiliser *la commande* ***N*** *qui opère de la même façon que* ***F*** *mais dans tout le fichier* (exécute autant de A et de W qu'il est nécessaire).

Il existe également une commande similaire à "**S**", c'est la juxtaposition "**J**" qui prend la forme :

J<chaîne 1>**↑Z**<chaîne 2>**↑Z**<chaîne 3>

On recherche d'abord la chaîne 1. Lorsque celle-ci est trouvée, on insère à la suite la chaîne 2 (sinon, il n'y a pas d'insertion). On recherche ensuite la chaîne 3, et on efface les caractères situés entre la fin de la chaîne 2 et le début de la chaîne 3 (si chaîne 3 n'existe pas, il n'y a pas d'effacement). C'est aussi simple que cela :

```
7: *B
1: *JFIN D^ZES 6 PREMIERS MOIS ^Z----
4: *OLT
4:  FIN DES 6 PREMIERS MOIS ----
4: *
```

Mais si l'on désire rechercher la fin de la ligne, terminée par les codes CR/LF (retour chariot/nouvelle ligne), comment représenter ces codes dans la chaîne à rechercher ? Cela aussi est prévu ! Il faut employer le code de contrôle **CTRL L** (voir tableau des codes de contrôle au chapitre 2).

ED nous permet aussi (quel luxe !) de répéter une ligne de commande donnée, autant de fois qu'on le souhaite. Dans ce cas, la ligne doit être précédée du caractère **M** (comme Macro-commande), lui-même précédé du nombre de fois désiré. Si le nombre est absent, la commande boucle jusqu'à rencontre d'une erreur :

```
                    ↑L = CR/LF
    4: *B
    1: *MF^L^ZI** ^Z

BREAK "#" AT ^Z
     : *B#T
    1:  MOIS DE L'ANNEE ----------------------------------------
    2:  ** janvier                  fevrier          mars
    3:  ** avril                    mai              juin
    4:  ** FIN DES 6 PREMIERS MOIS ----
    5:  ** juillet                  aout             septembre
    6:  ** octobre                  novembre         decembre
    7:  ** FIN DU SECOND SEMESTRE ----
    8:  ** *B
    1: *
```

La figure ci-dessus nous montre comment ajouter les caractères "**" au début de chaque ligne du fichier, et peut se traduire ainsi :

- chercher la fin de ligne (CTRL L)
- inclure ""**"
- boucler jusqu'à la fin du fichier.

Le message "BREAK" "#" nous indique que le nombre d'itérations demandé est trop grand (infini). Les autres types d'erreurs pouvant être rencontrés sont : "?" (commande inconnue), ">" (buffer mémoire plein), "0" (le fichier ".LIB" ne peut pas être ouvert).

Si vous trouvez que l'éditeur "travaille" trop vite et ne vous laisse pas le temps d'examiner chaque ligne affichée durant un listage, vous pouvez le faire "dormir"... en employant la commande "**Z**" à l'intérieur d'une macro-commande :

```
    1: *M2ZTL
    1:  MOIS DE L'ANNEE ------------------------------
    2:  ** janvier                fevrier          mars
    3:  ** avril                  mai              juin
    4:  ** FIN DES 6 PREMIERS MOIS ---
    5:  ** juillet                aout             septembre
    6:  ** octobre                novembre         decembre
    7:  ** FIN DU SECOND SEMESTRE ---
    8:  **
BREAK "#" AT Z
     : *
```

(vous avez aussi le droit d'aller dormir un peu, avant d'examiner la suite...)

PIP - second niveau

Nous avons vu précédemment que **PIP** était un programme permettant de généraliser les transferts entre fichiers et organes périphériques. En réalité, il est plus que cela, et il possède un certain nombre d'options très intéressantes que nous allons examiner maintenant, toujours au moyen d'exemples pratiques.

Avant toute chose, nous avons créé deux petits fichiers "outils" qui nous aideront à mieux saisir le mécanisme de certaines de ces commandes.

En voici la liste :

```
A>type fich1.txt
FICHIER 1 ----------
Ceci est un exemple
de fichier ASCII
        tabulation
A>type fich2.txt
FICHIER 2 ********
Voici un second exemple
de texte

A>
```

La tabulation à la dernière ligne du premier fichier, a été créée par l'éditeur au moyen du code **CTRL I** (toutes les 8 colonnes).

Voyons maintenant comment il est possible de fusionner (**MERGE** en anglais) deux fichiers :

```
A>pip x=fich1.txt,fich2.txt

A>type x
FICHIER 1 ----------
Ceci est un exemple
de fichier ASCII
        tabulation
FICHIER 2 ********
Voici un second exemple
de texte
```

La commande ci-dessus précise que le fichier de destination appelé "x" contiendra le fichier "fichl.txt" **et** le fichier "fich2.txt".

La liste du fichier "x" illustre cette fonction, ainsi que le "dump" suivant :

```
A>dump x

0000 46 49 43 48 49 45 52 20 31 20 2D 2D 2D 2D 2D 2D
0010 2D 2D 0D 0A 43 65 63 69 20 65 73 74 20 75 6E 20
0020 65 78 65 6D 70 6C 65 0D 0A 64 65 20 66 69 63 68
0030 69 65 72 20 41 53 43 49 49 0D 0A 09 74 61 62 75
0040 6C 61 74 69 6F 6E 0D 0A 46 49 43 48 49 45 52 20
0050 32 20 2A 2A 2A 2A 2A 2A 2A 2A 0D 0A 56 6F 69 63
0060 69 20 75 6E 20 73 65 63 6F 6E 64 20 65 78 65 6D
0070 70 6C 65 0D 0A 64 65 20 74 65 78 74 65 0D 0A 1A

A>
```

fich1.txt

fich2.txt

Nous voyons que les deux fichiers sont tout simplement juxtaposés.

Certains organes périphériques, tels que l'unité cassette ou le lecteur/perforateur de rubans, nécessitent une "amorce" en début et en fin de fichier. Ces blocs nuls permettent à l'appareil d'atteindre sa "vitesse de croisière" lors de la séquence de démarrage ou d'arrêt du moteur.

PIP nous permet de générer ces blocs en utilisant le mot-clé "NUL:" de la façon suivante :

```
A>pip x=nul:,fich1.txt,nul:

A>dump x

0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 46 49 43 48 49 45 52 20
0030 31 20 2D 2D 2D 2D 2D 2D 2D 2D 0D 0A 43 65 63 69
0040 20 65 73 74 20 75 6E 20 65 78 65 6D 70 6C 65 0D
0050 0A 64 65 20 66 69 63 68 69 65 72 20 41 53 43 49
0060 49 0D 0A 09 74 61 62 75 6C 61 74 69 6F 6E 0D 0A
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 1A 1A 1A 1A 1A 1A 1A 1A
00A0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
00B0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
00C0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
00D0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
00E0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
00F0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A

A>
```

40 x nuls

fich1.txt

40 x nuls

Le "dump" du fichier nous montre que 40 caractères nuls ont été ajoutés au début et à la fin du fichier. Dans notre exemple, nous avons utilisé le disque comme organe de sortie, de façon à pouvoir examiner le fichier.

Dans la pratique, la commande sera, par exemple :

```
PIP PUN:=NUL:,FICH1.TXT,NUL:
```

De la même manière, il est possible de générer une marque "fin de fichier" correspondant au code **CTRL Z**, en employant le mot-clé "EOF:". Exemple :

```
A>pip x=fich1.txt,eof:,fich2.txt

A>type x
FICHIER 1 ----------
Ceci est un exemple
de fichier ASCII
        tabulation
```

Arrêt sur ↑Z

Nous voyons que la liste du fichier se termine avant la fin physique, sur détection du code CTRL Z, et que tout ce qui suit est ignoré. Pourtant, notre fichier complet est bien là :

```
A>dump x

0000 46 49 43 48 49 45 52 20 31 20 2D 2D 2D 2D 2D 2D
0010 2D 2D 0D 0A 43 65 63 69 20 65 73 74 20 75 6E 20
0020 65 78 65 6D 70 6C 65 0D 0A 64 65 20 66 69 63 68
0030 69 65 72 20 41 53 43 49 49 0D 0A 09 74 61 62 75
0040 6C 61 74 69 6F 6E 0D 0A 1A 46 49 43 48 49 45 52
0050 20 32 20 2A 2A 2A 2A 2A 2A 2A 2A 0D 0A 56 6F 69
0060 63 69 20 75 6E 20 73 65 63 6F 6E 64 20 65 78 65
0070 6D 70 6C 65 0D 0A 64 65 20 74 65 78 74 65 0D 0A

A>
```

↑Z

PIP reconnait aussi deux organes logiques supplémentaires : **INP:** en entrée de caractères, et **OUT:** en sortie de caractères, que l'on peut utiliser ainsi :

```
PIP OUT:=fich1.txt
ou
PIP x=INP:
```

En standard, ces organes ne sont toutefois pas traités par CP/M et l'utilisateur doit modifier PIP pour qu'ils soient pris en compte.

Lorsque PIP rencontre le mot-clé "INP:" dans la commande, il effectue un branchement à l'adresse 103H. Normalement, on trouve une instruction "**RET**" (retour) à cette adresse, sauf si l'utilisateur a modifié cet endroit en conséquence pour lire un caractère sur un organe périphérique donné. Le caractère récupéré par cet appel est situé à l'adresse 109H (il est à la charge de l'utilisateur de le placer à cet endroit).

De même, lorsque PIP rencontre le mot-clé "OUT:" dans la commande, il effectue un branchement à l'adresse 106H, le caractère à émettre se trouvant dans le registre C du microprocesseur. Là

encore, en standard, nous trouvons une instruction de retour **RET**, comme le montre une petite incursion dans PIP :

```
A>
A>ddt pip.com
DDT VERS 2.2
NEXT  PC
1E00 0100
-l
  0100  JMP  04CE
  0103  RET
  0104  NOP
  0105  NOP
  0106  RET
  0107  NOP
  0108  NOP
  0109  LDAX D
  010A  NOP
  010B  NOP
  010C  NOP
-GO

A>
```

INP:

OUT:

La zone située de 109H à 1FFH est réservée à l'utilisateur pour exploiter ces deux routines d'émission et de réception de caractères. Notez que l'adresse 109 contient le code 1AH de l'EOF (correspondant à l'instruction LDAX D).

Il y a encore un autre organe logique connu de PIP : il s'agit de **PRN:** qui se comporte exactement de la même manière que **LST:** mais qui offre une "présentation" très attrayante de la liste (voir plus loin).

Le paramètre **B**, lorsqu'il est spécifié dans une commande PIP, force celui-ci à travailler en mode bloc. Cela signifie que les données lues seront placées dans un buffer mémoire avant d'être transmises au fichier de destination sur détection d'un EOF. Ce mode de transfert peut être nécessaire lorsque l'on utilise par exemple, une unité de cassette (lecture continue), sur laquelle l'ordinateur ne peut pas interrompre la lecture à n'importe quel moment.

En voulant lister le contenu d'un fichier ASCII sur l'imprimante ou la console, il peut être intéressant de "tronquer" les lignes affichées afin d'éviter une "ligne suite" qui viendra détruire toute la présentation du fichier à examiner. PIP permet cette fonction au moyen du paramètre **D** suivi d'un nombre de colonnes au-delà duquel la suite des caractères sera ignorée :

```
A>pip con:=fich1.txt[D5]
FICHI
Ceci
de fi
        tabul

A>
```

Notez une chose importante : tout paramètre ou ensemble de paramètres doit être donné entre "crochets" et doit immédiatement suivre la commande (pas d'espace).

En examinant la liste ci-dessus, on s'aperçoit que le code tabulation est ignoré par cette commande.

Un autre paramètre, **E** (comme Echo), permet de donner une trace sur la console des échanges se produisant entre deux fichiers ou deux organes périphériques :

```
A>pip x=fich1.txt[e]
FICHIER 1 ----------
Ceci est un exemple
de fichier ASCII
        tabulation

A>
```

Sans le paramètre **E**, le texte ci-dessus n'aurait pas été affiché. Mais que pensez-vous de ceci :

```
A>pip con:=fich1.txt[e]
FFIICCHHIIEERR  11  ----------------------

CCeeccii  eesstt  uunn  eexxeemmpplee

ddee  ffiicchhiieerr  AASSCCIIII

                ttaabbuullaattiioonn

A>
```

Vous vous souvenez du format .HEX ? C'est celui qui est généré par l'assembleur ASM pour représenter sous forme ASCII des fichiers binaires. Deux paramètres de PIP s'y rapportent : **H** et **I**.

En utilisant le premier lors d'un transfert de fichier HEX, PIP contrôle en même temps la validité des blocs échangés (calcul du checksum). Vous trouverez en appendice une description du format HEX.

```
A>pip con:=prog.hex[H]
:10010000113901 0E0ACD0500CD31013A3B01FE2324
:10011000CA0000213B013A3A015F1600193624114A
:100120003B010E09CD0500CD3101AF323A01C300CC
:0B013000011 16E010E09C30500320032
:03016E000D0A2453
:00010000FF

A>
```

En principe, tout fichier est terminé par une marque fin de fichier (prenant la forme :00 en format .HEX). Si l'on désire fusionner plusieurs fichiers HEX et les envoyer à un organe périphérique, il faut supprimer les EOF marquant la fin de chacun

d'eux. Cette fonction est réalisée par le paramètre **I**, qui exécute aussi, par la même occasion, la fonction du paramètre **H** :

```
A>pip con:=prog.hex[I]
:100100001139010E0ACD0500CD31013A3B01FE2324
:10011000CA0000213B013A3A015F160019362411 4A
:100120003B010E09CD0500CD3101AF323A01C300CC
:0B013000011 16E010E09C30500320032
:03016E000D0A2453

A>
```

Vous avez récupéré un fichier de texte en caractères majuscules, mais vous aimeriez le voir apparaître en minuscules sur votre console ? Le bon docteur PIP a aussi un remède pour cela (**L** comme Lower case = minuscules) :

```
A>pip con:=fich1.txt[L]
fichier 1 ----------
ceci est un exemple
de fichier ascii
        tabulation

A>
```

tout en minuscules

Notez le mot "fichier" initialement en majuscules.

Par contre, vous souhaiteriez avoir une trace écrite de ce fichier en caractères minuscules, mais votre imprimante ne "sait pas faire"... Alors imprimez-le en majuscules (**U** comme Upper case = majuscules) :

```
A>pip con:=fich1.txt[U]
FICHIER 1 ----------
CECI EST UN EXEMPLE
DE FICHIER ASCII
        TABULATION

A>
```

tout en majuscules

Il serait intéressant de numéroter les lignes de ce fichier: PIP le fera pour vous (**N** comme numérotation) :

```
pip con:=fich1.txt[N]
     1: FICHIER 1 ----------
     2: Ceci est un exemple
     3: de fichier ASCII
     4:          tabulation

A>
```

Revenons un peu en arrière, plus exactement au fichier "x" composé de "fich1.txt" et fich2.txt" séparés par un EOF.

```
A>pip x=fich1.txt,eof:,fich2.txt

A>pip con:=x
FICHIER 1 ----------
Ceci est un exemple
de fichier ASCII
        tabulation
A>
```

Nous ne voyons, souvenez-vous en, que la première partie du fichier. Pour voir la seconde partie, il suffit de spécifier le paramètre "O" (O comme Objet) qui permettra à PIP d'ignorer le code EOF :

```
A>pip con:=x[O]
FICHIER 1 ----------
Ceci est un exemple
de fichier ASCII
        tabulation
FICHIER 2 ********
Voici un second exemple
de texte

A>
```

le ↑Z est ignoré

En effet, lorsque l'on transmet, un fichier objet en binaire (.COM par exemple), il est fréquent de trouver des codes 1AH (LDAX D) correspondant à un EOF. Il est donc préférable, dans ce cas, de ne pas les supprimer du fichier.

Avec PIP, on peut aussi agir sur la présentation des fichiers transmis (à l'imprimante par exemple). Ainsi, il peut être souhaitable d'effectuer un saut de page (Form Feed = saut à la page suivante) avant de commencer l'impression :

```
A>pip x=fich1.txt[p1]

A>dump x

0000 0C 46 49 43 48 49 45 52 20 31 20 2D 2D 2D 2D 2D
0010 2D 2D 2D 0D 0A 43 65 63 69 20 65 73 74 20 75 6E
0020 20 65 78 65 6D 70 6C 65 0D 0A 64 65 20 66 69 63
0030 68 69 65 72 20 41 53 43 49 49 0D 0A 09 74 61 62
0040 75 6C 61 74 69 6F 6E 0D 0A 1A 1A 1A 1A 1A 1A 1A
0050 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0060 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0070 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A

A>
```

saut de page

Lorsque le paramètre P est seul ou suivi du nombre 1, le saut de page s'effectue toutes les 60 lignes, avec un saut au début du fichier, comme on peut le constater sur le "dump". Nous en reparlerons.

Dans l'immédiat, nous souhaiterions pouvoir abandonner automatiquement le listage d'un fichier lorsqu'une chaîne de caractères bien précise est rencontrée. PIP possède un paramètre (Q comme Quit) permettant cela, prenant la forme suivante :

Q<chaîne recherchée>↑Z

Essayons cela sur le mot "de" :

```
A>pip con:=fich1.txt[qde^Z]
FICHIER 1 ----------
Ceci est un exemple
de fichier ASCII
        tabulation

QUIT NOT FOUND: FICH1.TXT[QDE]
A>
```

Raté ! *Ca ne marche pas sur les codes minuscules*. Très bien. Essayons plutôt comme ceci :

```
A>PIP con:=fich1.txt[Qch^Z]
FICH

A>
```

Cette fois, il s'agit bien du résultat attendu. Mais comment faire avec les codes minuscules ? Il y a une solution : il faut lancer PIP par la seconde méthode d'appel (sans spécifier de commandes) :

```
A>PIP
*con:=fich1.txt[Qde^Z]
FICHIER 1 ------------
Ceci est un exemple
de
*
A>
```

Retour Chariot

En utilisant la première méthode, CP/M a la fâcheuse habitude de transcrire toutes les commandes en caractères majuscules. Dans la seconde méthode, nous passons la commande directement à PIP, et il n'y a donc pas de conversion... Il suffit de le savoir !

Mais nous pouvons également démarrer le transfert ou la liste, sur détection d'une chaîne donnée, au moyen du paramètre **S** (S comme Start = début) :

```
PIP con:=fich1.txt[S--^Z]
------------
Ceci est un exemple
de fichier ASCII
        tabulation

A>
```

ou même extraire une portion du fichier en combinant les deux paramètres :

```
A>PIP con:=fich1.txt[S1^Z QAS^Z]
1 ------------
Ceci est un exemple
de fichier AS

A>
```

Certaines imprimantes ne connaissent pas le code de tabulation (CTRL I) et l'ignorent tout bonnement. Lorsque l'on désire imprimer une liste d'assemblage, par exemple, c'est assez peu pratique.

Là encore, PIP vient à notre aide en transformant les codes de tabulation en caractères d'espace (expansion des codes de tabulation). Il suffit d'employer le paramètre **T** suivi de la longueur de la tabulation désirée :

```
A>pip x=fich1.txt[T4]

A>dump x

0000 46 49 43 48 49 45 52 20 31 20 2D 2D 2D 2D 2D 2D
0010 2D 2D 0D 0A 43 65 63 69 20 65 73 74 20 75 6E 20
0020 65 78 65 6D 70 6C 65 0D 0A 64 65 20 66 69 63 68
0030 69 65 72 20 41 53 43 49 49 0D 0A 20 20 20 20 74
0040 61 62 75 6C 61 74 69 6F 6E 0D 0A 1A 1A 1A 1A 1A
0050 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0060 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0070 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A

A>
```

Nous voyons sur le "dump" que le code 09H (tab) a été remplacé par 4 espaces (code 20H).

Lors d'une recopie de tous les fichiers d'une disquette (ou même d'un seul fichier particulièrement précieux), il peut être rassurant pour l'esprit de constater que la copie en question a été soigneusement contrôlée et vérifiée par PIP.

Celui-ci nous offre en effet un paramètre (**V** comme Vérification) qui, lorsqu'il est spécifié, force une relecture du fichier copié.

```
A>pip x=fich1.txt,fich2.txt[V]

A>
```

Nous avons vu au début du chapitre précédent, qu'un fichier pouvait être protégé contre la recopie ou la suppression, au moyen de la commande STAT. PIP nous permet d'ignorer cet état (R/O) si la commande comporte le paramètre **W**. Nous ne saurions trop attirer l'attention du lecteur sur le risque qu'il court en utilisant ce paramètre !

Par la même commande STAT, un fichier peut être rendu invisible (état SYS) et donc inaccessible en lecture par PIP. Le paramètre **R** (Read) permet également - mais sans risque cette fois - d'outrepasser cette interdiction. Ces deux paramètres (W et R) ne sont présents qu'en version 2 de CP/M.

Revenons au paramètre **P** qui permet d'insérer des codes de saut de page dans un fichier, et utilisons celui-ci avec une valeur supérieure à 1 :

```
A>pip x=fich1.txt[P2]

A>dump x

0000 0C 46 49 43 48 49 45 52 20 31 20 2D 2D 2D 2D 2D
0010 2D 2D 2D 0D 0A 43 65 63 69 20 65 73 74 20 75 6E
0020 20 65 78 65 6D 70 6C 65 0D 0A 0C 64 65 20 66 69
0030 63 68 69 65 72 20 41 53 43 49 49 0D 0A 09 74 61
0040 62 75 6C 61 74 69 6F 6E 0D 0A 0C 1A 1A 1A 1A 1A
0050 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0060 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0070 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
A>
```

A l'inverse, le paramètre **F** supprime les codes de saut de page dans le fichier, ce qui peut être utile sur certaines imprimantes qui font "n'importe quoi" lorsqu'elles rencontrent ce code :

```
A>pip toto=x[F]

A>dump toto

0000 46 49 43 48 49 45 52 20 31 20 2D 2D 2D 2D 2D 2D
0010 2D 2D 0D 0A 43 65 63 69 20 65 73 74 20 75 6E 20
0020 65 78 65 6D 70 6C 65 0D 0A 64 65 20 66 69 63 68
0030 69 65 72 20 41 53 43 49 49 0D 0A 09 74 61 62 75
0040 6C 61 74 69 6F 6E 0D 0A 1A 1A 1A 1A 1A 1A 1A 1A
0050 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0060 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0070 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A

A>
```

Il nous manque un paramètre : **Z.** Ce sera le dernier. Certains périphériques génèrent les caractères ASCII accompagnés d'un bit de parité (poids fort à 1 ou à 0 selon que la somme des bits de l'octet est paire ou impaire). Ce contrôle pourrait être bien pratique, mais il manque de standardisation (parité paire ? parité impaire ?) et occasionne souvent quelques problèmes de dialogue. Avec le paramètre **Z** de PIP, le bit de parité est forcé systématiquement à zéro sur tous les caractères reçus du fichier ou de l'organe d'entrée.

Tous les codes que nous venons d'examiner, peuvent, bien sûr, être cumulés dans une même commande : **T8NP** réalise les fonctions suivantes :

- Expansion des codes tabulation à 8 espaces
- Numérotation des lignes du fichier
- Saut de page toutes les 60 lignes

Mais pour réaliser cela automatiquement sur l'imprimante, il suffit de s'adresser à l'organe logique **PRN:** créé spécialement dans ce but :

```
PIP PRN:=fichl.txt
```

Comme nous venons de le voir, PIP est bien plus qu'un utilitaire réalisant de vulgaires copies de fichiers. C'est l'une des commandes maîtresses de CP/M, offrant une grande souplesse à l'utilisateur.

DDT - second niveau

Au chapitre précédent, nous avions déjà examiné la plupart des commandes de l'utilitaire de mise au point : **DDT.** Il nous reste à faire quelques approfondissements, et pour cela, nous nous aiderons du programme PROG créé précédemment.

```
ddt
DDT VERS 2.2
-Iprog.hex
-R100
NEXT  PC
0271 0100
-
```

Les commandes ci-dessus ne devraient pas vous étonner, mise à part le "R100"... qui appelle des compléments d'informations. Mais avant tout, aviez-vous constaté que **DDT** était aussi capable de charger des programmes au format HEX (commandes **I** et **R**) ?

La commande I permet de *définir le nom du programme* à charger, alors que la **commande R** *appelle ce programme en mémoire*. Lorsque cette dernière est suivie d'une valeur numérique (en hexa), le chargement en mémoire, qui s'effectue habituellement en 100H, est repoussé du nombre d'octets spécifié par cette valeur. Dans notre cas, le programme sera chargé à l'adresse 200H. Le paramètre suivant **R** n'est donc pas une adresse mais une valeur relative à 100H.

La preuve, c'est que notre programme n'a pas été chargé en 100H :

```
-L100,107
  0100  LXI  B,0FBC
  0103  JMP  013D
  0106  MOV  B,E
  0107  MOV  C,A
  0108
-
```

où l'on trouve n'importe quoi..., mais en 200H :

```
-L200
  0200  LXI  D,0139
  0203  MVI  C,0A
  0205  CALL 0005
  0208  CALL 0131
  020B  LDA  013B
  020E  CPI  23
  0210  JZ   0000
  0213  LXI  H,013B
  0216  LDA  013A
  0219  MOV  E,A
  021A  MVI  D,00
-
```

Malheureusement, il a été assemblé avec 100H comme origine (ORG 100H avec ASM), et il n'est donc pas exécutable à cet emplacement (à l'adresse 213, le LXI H,138 devrait être : LXI H,238... Et il n'est pas le seul à être dans ce cas !).

Cette commande est donc réservée à des cas particuliers, par exemple, lorsque l'on désire charger un programme (assemblé avec un ORG en conséquence) sans détruire celui qui est situé au début de la zone TPA.

Bref... il va falloir tout recharger ! A moins que nous utilisions la **commande M** de DDT, qui permet justement de *déplacer une zone mémoire*. Elle est de la forme :

M<adresse début>,<adresse fin>,<adresse destination>

Dans notre cas, ce sera :

```
-M200,271,100
-L100
  0100  LXI  D,0139
  0103  MVI  C,0A
  0105  CALL 0005
  0108  CALL 0131
  010B  LDA  013B
  010E  CPI  23
  0110  JZ   0000
  0113  LXI  H,013B
  0116  LDA  013A
  0119  MOV  E,A
  011A  MVI  D,00
-
```

Et cette fois, la liste symbolique nous confirme bien que notre programme est à sa bonne place. Toutefois, il existe toujours à son ancienne adresse. Pas pour longtemps... :

```
-F200,271,ff
-D200,271
0200 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0210 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0220 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0230 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0240 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0250 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0260 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0270 FF FF ..
-
```

La commande F (Fill=remplissage), *permet de remplir une zone mémoire avec une valeur donnée*. Elle est de la forme :

F<adresse début>,<adresse fin>,<valeur>

Le "dump" ci-dessous montre que la zone 200 à 271 (hexa) est remplie de codes FF.

Reportons-nous au listing d'assemblage de PROG. A l'adresse 11FH, nous y voyons un "lxi d,texte" (code généré :11 3B 01). Nous avons l'intention d'ajouter une petite "verrue" à notre programme, de façon à ce que celui-ci ne copie plus fidèlement chaque caractère de la phrase entrée au clavier mais le caractère suivant (que ça va être drôle !). Cette verrue sera placée à l'adresse libre (NEXT) donnée par DDT (soit : 171H), et le programme sera "intercepté" à l'adresse 11F, juste avant l'affichage de la phrase.

```
        mvi     m,'$'           ; fin texte par $
;
                                JMP VERRUE
        lxi     d,texte         ; pointe debut texte
SUITE   mvi     c,print         ; code sortie
        call    bdos            ; appel bdos
;
        call    ligne           ; envoi cr/lf
;
        xra     a               ; 0---> nbre carac recus
        sta     ncr             ;
        jmp     prog            ; suite
;
ligne   lxi     d,crlf          ; routine envoi cr/lf
        mvi     c,print         ; code sortie
        jmp     bdos            ; apeel bdos + retour
;
; zones des donnees ------------------------
;
bufrec  db      50      ; longueur buffer
ncr     db      0       ; nbre carac recus
texte   ds      51      ; buffer texte
crlf    db      0dh,0ah,'$'
        end     prog
```

```
VERRUE  LXI   H,TEXTE
        MOV   A,M
        CPI   '$'
        JZ    FIN
        INR   A
        MOV   M,A
        INX   H
        JMP   ENCORE
FIN     LXI   D,TEXTE
        JMP   SUITE
```

Nous avons déjà appris à faire des modifications dans un programme, à l'aide de la commande de substitution (**S**) :

```
-S11f
011F 11 c3
0120 3B 71
0121 01 .
-
```

JMP VERRUE

Mais il y a un moyen beaucoup plus simple : utiliser la commande d'assemblage (**A**) de DDT, en spécifiant l'adresse de départ où l'on désire assembler :

```
-A171
0171  lxi       h,texte
?
-
```

c'est pas malin!

Bon. C'était une blague ! Il ne faut tout de même pas trop en demander...! DDT n'accepte pas les symboles autres que ceux des instructions, ce qui n'est déjà pas si mal ! Recommençons :

```
-A171
0171  lxi   h,13b
0174  mov   a,m
0175  cpi   24
0177  jz    180
017A  inr   a
017B  mov   m,a
017C  inh   h
?
017C  inx   h
017D  jmp   174
0180  lxi   d,13b
0183  jmp   122
0186
-
```

Seulement les instructions !

Que pensez-vous de notre verrue ? Malicieuse hein ? Remarquez que le "lxi d,13b" qui avait été remplacé par le JMP à l'adresse 11F doit être restitué (adresse 122).

Lançons donc ce programme modifié, mais arrêtons-nous toutefois au début de la verrue en plaçant un point d'arrêt à l'adresse 171 :

```
-G100,171
abcd
*0171
-
```

Le message "abcd" est entré au clavier et, dès la frappe du "retour chariot", DDT affiche l'adresse d'arrêt précédée d'un "*".

Là, nous pouvons examiner le contenu des registres (commande **X**), et faire du pas-à-pas instruction (commande **T**) :

```
-x
COZOMOEOIO A=04 B=0070 D=0004 H=013F S=0100 P=0171 LXI  H,013B
-T
COZOMOEOIO A=04 B=0070 D=0004 H=013F S=0100 P=0171 LXI  H,013B*0174
-T
COZOMOEOIO A=04 B=0070 D=0004 H=013B S=0100 P=0174 MOV  A,M*0175
-T8
COZOMOEOIO A=61 B=0070 D=0004 H=013B S=0100 P=0175 CPI  24
COZOMOEOIO A=61 B=0070 D=0004 H=013B S=0100 P=0177 JZ   0180
COZOMOEOIO A=61 B=0070 D=0004 H=013B S=0100 P=017A INR  A
COZOMOEOIO A=62 B=0070 D=0004 H=013B S=0100 P=017B MOV  M,A
COZOMOEOIO A=62 B=0070 D=0004 H=013B S=0100 P=017C INX  H
COZOMOEOIO A=62 B=0070 D=0004 H=013C S=0100 P=017D JMP  0174
COZOMOEOIO A=62 B=0070 D=0004 H=013C S=0100 P=0174 MOV  A,M
COZOMOEOIO A=62 B=0070 D=0004 H=013C S=0100 P=0175 CPI  24*0177
-
```

Nous pouvons aussi exécuter un groupe d'instructions (8 par exemple), en "traçant" uniquement la dernière, au moyen de la commande **U** :

```
-U8
COZOMOEOIO A=62 B=0070 D=0004 H=013C S=0100 P=0177 JZ   0180*017A
-U8
COZOMOE1IO A=63 B=0070 D=0004 H=013D S=0100 P=017A INR  A*017B
-
```

Pour repartir à l'adresse courante, s'arrêter en 180 (par exemple), "tracer" deux instructions et repartir avec un prochain arrêt en 12a, nous ferons :

```
-G,180
*0180
-X
COZ1MOE1I1 A=24 B=0070 D=0004 H=013F S=0100 P=0180 LXI  D,013B
-T
COZ1MOE1I1 A=24 B=0070 D=0004 H=013F S=0100 P=0180 LXI  D,013B*0183
-T
COZ1MOE1I1 A=24 B=0070 D=013B H=013F S=0100 P=0183 JMP  0122*0122
-G,12a
bcde
*012A
-
```

La verrue a rempli sa fonction : le message "bcde" a été affiché !

Remarquez la virgule après le "G". Si l'adresse de lancement n'est pas spécifiée, c'est la valeur courante de PC (compteur programme) qui est prise par défaut.

On peut aussi spécifier deux points d'arrêt, si l'on hésite entre deux valeurs d'adresses :

```
-G,100,171
*0100
-
```

Dès le passage sur l'une des deux valeurs, les points d'arrêt seront annulés. Attention à ne pas se tromper : si le programme ne passe jamais sur la ou les valeurs spécifiées, le retour à DDT est impossible !

Lorsque plus de deux points d'arrêt doivent être spécifiés, il est toujours possible de remplacer l'instruction sur laquelle on veut s'arrêter par "FF" (il faudra penser à la restituer par la suite...), qui correspond au code de l'instruction RST 7. Lorsque le programme rencontre celle-ci, un retour à DDT est automatiquement effectué (à condition que ce dernier soit chargé).

```
-S171
0171 21 ff
0172 3B .
-g
au clair de la lune
*0171
-x
COZOMOE010 A=13 B=0070 D=0013 H=014E S=0100 P=0171 RST  07
-
```

Restituons l'instruction primitive et voyons le résultat :

```
S171
0171 FF 21
0172 3B .
-s171,100
bv!dmbjs!ef!mb!mvof
*0100
-
```

(ce serait drôle d'écrire tout un livre comme cela !)

L'instruction "**X**" de DDT ne permet pas seulement de visualiser les registres du microprocesseur. Elle autorise aussi leur altération :

```
-Xa
A=00 ff
-x
COZ1MOE1IO A=FF B=0070 D=3EF8 H=0000 S=0100 P=0100 LXI  D,0139
-
```

Le registre A a été forcé avec la valeur "FF". En ce qui concerne les indicateurs, ils peuvent aussi être modifiés :

```
-xz
Z1 0
-x
COZOMOE1IO A=FF B=0070 D=3EF8 H=0000 S=0100 P=0100 LXI  D,0139
-
```

Dans l'exemple ci-dessus, DDT affiche la valeur de l'indicateur Z (affichage de : Z1, donc Z=1), et attend une réponse. En frappant "retour chariot", l'indicateur conserve sa valeur primitive. En frappant 0 ou 1, la valeur est remplacée.

On peut aussi forcer le compteur programme (bien que ce registre ne soit pas directement accessible au programmeur), d'une autre façon qu'avec la commande **G** :

```
-xp
P=0100 171
-t
COZOMOE1IO A=FF B=0070 D=3EF8 H=0000 S=0100 P=0171 LXI  H,013B*0174
-t
COZOMOE1IO A=FF B=0070 D=3EF8 H=013B S=0100 P=0174 MOV  A,M*0175
-
```

Enfin, DDT vous aide dans vos démêlés avec la *base 16* : la **commande H** *suivie de deux valeurs (hexa) en délivre la somme et la différence :*

```
H12ff,a4
13A3 125B
-
```

(à notre avis, cela ne vaut pas la base 0, qui est beaucoup plus reposante...)

```
-GO

A>
```

Si vous souhaitez conserver le programme modifié (celui-ci en particulier, n'a aucun intérêt) n'oubliez pas la commande **SAVE.**

ANNEXES

ETES-VOUS BIEN ARRIVES ?

La course est terminée. Cette course là tout au moins, mais nous espérons que vous en ferez bien d'autres ! Si vous avez pris un mauvais départ, ce n'est pas grave (il n'y a personne pour vous "siffler"...) : recommencez ! Et cette fois-ci, c'est sûr, vous irez un peu plus loin.

Jetez un coup d'oeil dans les appendices, révisez les tableaux, n'hésitez pas à consulter d'autres ouvrages si nécessaire.

Ne vous arrêtez pas non plus, sur un petit détail : chaque chose vient en son temps, et ce que vous n'avez pas compris aujourd'hui, vous le comprendrez demain (ou peut-être cette nuit même ?) à moins que ce ne soit dans quelques années...

Mais qu'importe, cela ne doit pas être un prétexte pour s'arrêter en si bon chemin !

AFFECTATION MEMOIRE

La **page zéro** de la mémoire CP/M est formée de 256 octets (adresse 0 à FFH) dont voici la description :

Adresse	Description
0 a 2	Branchement demarrage a froid du BIOS
3	Octet entree/sortie (IOBYTE)
4	Numero disque courant et zone utilisateur
5 a 7	Branchement au BDOS
8 a 37H	zone reservee (vecteurs d'interruptions)
38H a 3AH	RST7 (utilise par DDT)
3BH a 3FH	reserve (vecteur interruption)
40H a 4FH	zone tampon utilisee par BIOS
50H a 5BH	non utilise
5CH a 7FH	zone par defaut du FCB
80H a FFH	zone tampon DMA pour entree/sortie disque

Les zones suivantes dépendant de la version de CP/M et de la configuration utilisée, ne sont données qu'à titre indicatif.

Dans les adresses qui suivent, "**x**" représente la taille mémoire maximale du système moins 20Ko en version 2 ou moins 16Ko en version 1.4.

Nom de zone	Adresse debut	Adresse fin
TPA (v2)	100H	33FFH+x
(v1)	100H	28FFH+x
CCP (v2)	3400H+x	3BFFH+x
(v1)	2900H+x	30FFH+x
BDOS (v2)	3C00H+x	49FFH+x
(v1)	3100H+x	3DFFH+x
BIOS (v2)	4A00H+x	4FFFH+x
(v1)	3E00H+x	3FFFH+x

DESCRIPTION DE L'OCTET ENTREE/SORTIE

Cet octet (nommé **IOBYTE**), est situé à l'adresse 3 de la mémoire, et contient les affectations des organes périphériques standards connus de CP/M. En voici la description :

bit	7	6	5	4	3	2	1	0
	LST:		PUN:		RDR:		CON:	
	00 : TTY:		00 : TTY:		00 : TTY:		00 : TTY:	
	01 : CRT:		01 : PTP:		01 : PTR:		01 : CRT:	
	10 : LPT:		10 : UP1:		10 : UR1:		10 : BAT:	
	11 : UL1:		11 : UP2:		11 : UR2:		11 : UC1:	

Signification des organes périphériques :

TTY: Télétype
CRT: Ecran de visualisation
BAT: Train de travaux (RDR en entrée, LST en sortie)
UC1: Console définie par l'utilisateur
LPT: Imprimante
UL1: Sortie de listes définie par l'utilisateur
PTR: Lecteur ruban perforé
UR1 et **UR2:** organes d'entrée (utilisateur)
PTP: Perforateur de rubans
UP1 et **UP2:** organes de sortie (utilisateur)

Prenons un exemple. Si l'imprimante est affectée à **LST:**, l'écran vidéo à la console (**CON:**) et la Télétype aux entrées/sorties rubans (**PUN:** et **RDR:**), le codage de l'octet sera le suivant :

10 00 00 01

soit la valeur 81H.

POINTS D'ENTREES BDOS

CP/M offre à l'utilisateur écrivant ses programmes en assembleur, un choix de 39 fonctions dont la séquence d'appel est la suivante :

```
LXI  D,<parametre>
MVI  C,<numero de fonction>
CALL 5
```

Le programme d'affichage d'une ligne du chapitre 3 utilisait deux de ces fonctions.

Le CALL 5 correspond au point d'entrée dans le BDOS. Lorsqu'une valeur de 16 bits doit être passée à la routine, le double registre DE est utilisé. Si une valeur de 8 bits seulement doit être passée, seul le registre E est utilisé.

Dans le cas des fonctions disquette, 16 bits sont insuffisants pour passer tous les paramètres utiles à la fonction. DE pointera donc sur une table appelée FCB (File Control Bloc = Bloc de Contrôle du Fichier) décrite en appendice.

Le registre C contient le numéro correspondant à la fonction devant être réalisée. Cette fonction peut être définie préalablement par EQU :

```
BDOS      EQU    5      ; point entree BDOS
ENTREE    EQU    1      ; console input
SORTIE    EQU    2      ; console output
          ....
          MVI    C,ENTREE
          CALL   BDOS
          ....
```

Dans le cas où la fonction fournit une valeur de retour, celle-ci se trouvera dans le registre A si la valeur est sur 8 bits, ou dans le double registre HL si la valeur est sur 16 bits.

Attention : il est à la charge de l'utilisateur de sauvegarder tous ses registres avant appel d'une fonction, ceux-ci risquant d'être détruits au retour.

Nous ne rentrerons pas dans le détail de chacune de ces fonctions, mais nous nous bornerons simplement à en donner la liste.

Fonction	No (C)	Parametre en entree	Parametre en sortie
System Reset	00	//////////////	///////////////
Console Input	01	//////////////	A=caractere
Console Output	02	E=caractere	///////////////
Reader Input	03	//////////////	A=caractere
Punch Output	04	E=caractere	///////////////
List Output	05	E=caractere	///////////////
Direct console I/O	06	E=0FFH (inp)	A=caractere
		E=cara (out)	A=status
Get I/O byte	07	//////////////	A=iobyte
Set I/O byte	08	E=iobyte	///////////////
Print String	09	DE=adr chaine	///////////////
Read console buffer	10	DE=adr buffer	buffer
Get console status	11	//////////////	A=0FFH (pret)
		//////////////	A=0 (pas pret)
Return Version #	12	//////////////	HL=no version
Reset Disk System	13	//////////////	///////////////
Select Disk	14	E=num disque	///////////////
Open File	15	DE=adr FCB	A=0FFH: absent
Close File	16	DE=adr FCB	A=0FFH: absent
Search for first	17	DE=adr FCB	A=0FFH: absent
Search for next	18	//////////////	A=0FFH: absent
delete File	19	DE=adr FCB	A=0FFH: absent
Read Sequential	20	DE=adr FCB	A=0: correct
			A<>0: fin fich
Write Sequent	21	DE=adr FCB	A=0: correct
			A<>0:disq plein
Create File	22	DE=adr FCB	A=0FFH:catalo-
			gue plein
Rename File	23	DE=adr FCB	A=0FFH: absent
Return Login vector	24	//////////////	HL=Login vector
Return Current disk	25	//////////////	A=num disque
Set DMA Address	26	DE=adr DMA	///////////////
Get Address (Alloc)	27	//////////////	HL=adr alloc
Write Protect Disk	28	//////////////	///////////////
Get R/O Vector	29	//////////////	HL=R/O vector
Set File Attrib	30	DE=adr FCB	///////////////
Get Addr(disk parm)	31	//////////////	HL=adr DPBASE
Set/Get User Code	32	E=0FFH:lect	A=code user
		E=code:ecrit	
Read Random	33	DE=adr FCB	A=code erreur
Write Random	34	DE=adr FCB	A=code erreur
Compute File Size	35	DE=adr FCB	FCB r0,r1,r2
Set Random Record	36	DE=adr FCB	FCB r0,r1,r2
Reset Drive	37	DE=num drive	A=0
Write Random with zero fill	40	DE=adr FCB	A=code erreur

POINTS D'ENTREES BIOS

De la même façon que pour le BDOS, il est possible d'atteindre certaines routines du **BIOS** (elles-mêmes utilisées par le BDOS). Le BIOS n'étant pas situé à une adresse fixe, l'utilisateur trouve son adresse absolue aux emplacements 1 et 2 de la page 0.

Pour effectuer les fonctions suivantes, il faudra ajouter cette valeur à l'adresse donnée dans le tableau.

Adresse +(BIOS)	Fonction	Description
00	BOOT	Demarrage a froid
03	WBOOT	Demarrage a chaud
06	CONST	Test si console prete (A=FF si ok)
09	CONIN	Lecture console (A=caractere)
0CH	CONOUT	Sortie console (C=caractere)
0FH	LIST	Sortie liste (C=caractere)
12H	PUNCH	Sortie perfo (C=caractere)
15H	READER	Lecture ruban (A=caractere)
18H	HOME	Tete disque en piste 0
1BH	SELDSK	Selection disque (C=num disque)
1EH	SETTRK	Position piste (C=num piste)
21H	SETSEC	Position secteur (C=num secteur)
24H	SETDMA	Definition adr tampon (BC=ad dma)
27H	READ	Lecture secteur (A=retour status)
2AH	WRITE	Ecriture secteur
2DH	LISTST	Test status LST: (A=FF si ok)
30H	SECTRAN	Conversion numero secteur logique en numero secteur physique

DESCRIPTION DU FCB (File Control Bloc)

Le **FCB** ou **bloc de contrôle fichier**, est une zone mémoire de 36 octets servant à communiquer avec un fichier disquette.

Lors de l'appel des fonctions disquette du BDOS, le double registre DE doit pointer au début de cette zone. Voici sa description :

```
            +----+-----------------------------------+
FCB -----> dr ! f1   f2   f3   f4   f5   f6   f7 !
            +----+----+----+----+----+----+----+----+
FCB+8 ---> f8 ! t1   t2   t3 ! ex ! s1 ! s2 ! rc !
            +----+----+----+----+----+----+----+----+
FCB+16 --> d0   d1   d2   d3   d4   d5   d6   d7 !
            +----+----+----+----+----+----+----+----+
FCB+24 --> d8   d9  d10  d11  d12  d13  d14  d15 !
            +----+----+----+----+----+----+----+----+
FCB+32 --> cr ! r0   r1   r2 !
            +----+----+----+----+
```

- **dr** est le numéro de disquette :

  0 --> unité implicite
  1 --> unité A
  2 --> unité B

- **f1** à **f8** est le nom du fichier complété éventuellement par des espaces.

- **t1** à **t3** est le type de fichier. Ce champ contient aussi les attributs :

  bit poids fort de t1 = 1 --> fichier R/O
  bit poids fort de t2 = 1 --> fichier SYS

- **ex** est le numéro d'extension courante.

- **s1** et **s2** sont réservés au système.

- **rc** désigne le nombre d'articles (0 à 127) dans l'extension **ex**.

- **d0** à **d15** sont les adresses des blocs utilisés pour stocker le programme.

- **cr** est le numéro d'article auquel on veut accéder en séquentiel.

- **r0** à **r2** correspondent au numéro d'article auquel on veut accéder en mode direct (**r0** est le poids faible de ce numéro).

OCCUPATION DISQUETTE

Elle est variable selon le type de support employé : disquette ou minidisquette, simple ou double densité, etc... Voici une disposition très fréquente, rencontrée sur une disquette simple densité (77 pistes de 26 secteurs) :

Piste / secteur	Contenu
00 / 01	Boot
00 / 02 a 00 / 17	CCP
00 / 18 a 01 / 19	BDOS
01 / 20 a 01 / 26	BIOS
02 / 01 a 76 / 26	Catalogue et fichiers

FORMAT HEX INTEL

Le fichier au format HEX est formé de blocs séquentiels dans lesquels on distingue : les blocs d'informations et les blocs de fin de fichiers. Entre ces blocs, on trouve des zones inter-blocs composées généralement de caractères nuls.

Chaque bloc est composé de six zones :

```
///+-----+-----+-----------+-----+---------//-+-----+///
///!  :  ! LG  !    ADR    ! REC !    DATA    ! CS  !///
///+--+--+--+--+--+--+--+--+--+--+---------//-+--+--+///
```

- Le caractère ":" est l'indicateur du début de bloc. Tant que le programme de lecture ne trouve pas ce caractère, il n'entreprend aucun traitement.

- **LG** indique la longueur du bloc d'informations (DATA) exprimée en hexadécimal. En général, les blocs sont d'une longueur de 16 caractères (10H). Si la longueur est nulle (00), le bloc est un bloc de fin de fichier et le champ DATA n'existe pas.

- **ADR** est le champ "adresse d'implantation" exprimé en hexadécimal. C'est à cette adresse que seront implantées en mémoire les informations contenues dans le champ DATA.

- **REC** spécifie le type de bloc auquel on a affaire : 00 désigne un bloc d'informations ou un bloc fin de fichier, alors que 01 désigne toujours un bloc fin de fichier.

- **DATA** représente le champ contenant les informations codées en hexadécimal.

- **CS** est le champ de contrôle du bloc (checksum). La somme des octets des champs LG + ADR + REC + DATA + CS est telle que l'octet de poids faible de cette somme doit être 00 si le contrôle est correct.

Prenons un exemple de bloc :

:	03	016E	00	0D 0A 24	53
	LG	ADR	REC	DATA	CS

La longueur du champ DATA est de 3 octets, l'adresse d'implantation en mémoire est 16EH, le type de bloc est 0, les 3 octets d'informations sont 0D, 0A et 24, et l'octet de checksum est 53. Cette valeur est-elle exacte ?

```
LG      03
ADR     01
        6E
REC     00
DATA    0D
        0A
        24
CS      53
      ------
      0100
```

L'octet de poids faible est 00 : le bloc est correct.

En général, le bloc fin prend la forme suivante :

: 00 ADR 00 CS

dans laquelle ADR représente l'adresse de lancement du programme contenu dans le fichier HEX. Pour l'adresse 0100H, la valeur de CS sera donc 0FFH.

L'ASSEMBLEUR ASM

L'assembleur dont nous allons parler ci-dessous est écrit pour le microprocesseur 8080 d'Intel. C'est celui qui est fourni en standard avec le système CP/M pour ordinateurs 8 bits.

Format des lignes assembleur

Chaque ligne est formée de cinq champs :

- un *numéro de ligne*
- un *label* de 1 à 16 caractères, éventuellement terminé par le signe ":"
- un *nom d'opération* (code instruction)
- un *opérande* (éventuel)
- un *commentaire* débutant par le caractère ";"

Bien que la ligne soit normalement terminée par le code "retour chariot", on peut placer plusieurs lignes logiques dans une ligne physique si l'on utilise le séparateur "!".

Exemple :

```
POP H ! POP B ! POP D
```

est équivalent à :

```
POP H
POP B
POP D
```

Les noms de labels peuvent utiliser le signe "$" qui, bien qu'ignoré par l'assembleur, permet une meilleure interprétation.

Exemple :

prixdevente est plus lisible que prixdevente

mais possède la même signification.

Les labels réservés (donc à ne pas employer) sont :

A, B, C, D, E, H, L, M, SP, PSW

et désignent les registres du microprocesseur 8080.

Constantes

Les constantes peuvent être exprimées dans plusieurs systèmes de numération, lorsqu'elles sont terminées par :

B	Binaire	Exemple : 1101B
O ou Q	Octel (base 8)	Exemple : 37Q
D (rien)	Décimal	Exemple : 123 ou 123D
H	Hexadécimal (base 16)	Exemple : 0FFH

Les chaînes de caractères sont définies entre apostrophes :

'programme' ou 'liste des instructions'

Opérateurs

Onze opérateurs peuvent être employés en zone opérande des instructions. Ce sont :

+	addition
-	soustraction
*	multiplication
/	division (entiere)
MOD	Module (reste de la division entiere)
NOT	Complement logique
AND	Intersection logique
OR	Reunion logique
XOR	Disjonction
SHL	Decalage gauche de 1 bit
SHR	Decalage droit de 1 bit

Registres microprocesseur

Dix registres sont utilisés par les instructions du langage machine. Selon les opérations, ils peuvent être considérés comme simples registres (8 bits) ou doubles registres (16 bits). Ces derniers sont notés entre parenthèses ci-dessous.

```
+---------------+---------------+
!       A       !  indicateurs  !   (PSW)
+---------------+---------------+
!       B       !       C       !   (B)
+---------------+---------------+
!       D       !       E       !   (D)
+---------------+---------------+
!       H       !       L       !   (M)
+---------------+---------------+
!       Compteur programme      !   (PC)
+-------------------------------+
!        pointeur de pile       !   (SP)
+-------------------------------+
```

Les indicateurs, contenus dans un simple registre, sont positionnés à la suite de certaines instructions.

```
+-----+-----+-----+-----+-----+-----+-----+-----+
!  S  !  Z  !//////!  AC !//////!  P  !//////!  C  !
+-----+-----+-----+-----+-----+-----+-----+-----+
   V     V           V           V           V
   !     !           !           !           +--> Carry
   !     !           !           +--------------> Parity
   !     !           +--------------------------> Aux carry
   !     +-------------------------------------> Zero
   +-----------------------------------------------> Sign
```

Directives

Les directives (ou pseudo-instructions) connues de l'assembleur ASM sont au nombre de huit :

ORG	Fixation d'une adresse d'origine
END	Indication de la fin de programme
EQU	Determination d'equivalence (definitive)
SET	Affectation d'une valeur (modifiable)
IF	Assemblage conditionnel
...	
ENDIF	Fin d'une sequence conditionnelle
DB	Definition octet
DW	Definition mot
DS	Reservation d'octets

Codes opérations

Signification des expressions employées dans les tableaux suivants :

<dr> : doubles registres B, D, H, SP
<r1>, <r2>, <reg> : simples registres A, B, C, D, E, H, L ou M
<pr> : doubles registres B, D, H, PSW
<port> : adresse de port de 0 a 0FFH
adr : adresse memoire 16 bits
v16 : valeur 16 bits (mot)
v8 : valeur 8 bits (octets)

. Les instructions de saut

C3	JMP	adr	Saut inconditionnel a adr
	JNZ	adr	Saut si Z non vrai
CA	JZ	adr	Saut si Z vrai
	JNC	adr	Saut si C non vrai
	JC	adr	Saut si C vrai
	JPO	adr	Saut si parite impaire (PO)
	JPE	adr	Saut si parite paire (PE)
	JP	adr	Saut si resultat positif
	JM	adr	Saut si resultat negatif

. Appels et retours de sous-programmes

CD	CALL	adr	Appel inconditionnel
	CNZ	adr	Appel si Z non vrai
	CZ	adr	Appel si Z vrai
	CNC	adr	Appel si C non vrai
	CC	adr	Appel si C vrai
	CPO	adr	Appel si parite impaire
	CPE	adr	Appel si parite paire
	CP	adr	Appel si resultat positif
	CM	adr	Appel si resultat negatif
	RST	<0-7>	Appel court (1 octet)
C9	RET		Retour sous-programme
	RNZ		Retour si Z non vrai
	RZ		Retour si Z vrai
	RNC		Retour si C non vrai
	RC		Retour si C vrai
	RPO		Retour si parite impaire
	RPE		Retour si parite paire
	RP		Retour si resultat positif
	RM		Retour si resultat negatif

. Chargements immédiats

MVI <reg>,<v8>	<v8> --> simple registre <reg>
ADI <v8>	Ajoute <v8> a A
ACI <v8>	Ajoute <v8>+Carry a A
SUI <v8>	Soustrait <v8> a A
SBI <v8>	Soustrait <v8>+Carry a A
ANI <v8>	ET logique entre <v8> et A dans A
XRI <v8>	OU exclusif entre <v8> et A dans A
ORI <v8>	OU logique entre <v8> et A dans A
CPI <v8>	Compare A et <v8>
LXI <dr>,<v16>	Charge le double reg. avec <v16>

. Incréments et décréments

INR <reg>	Increment simple registre
DCR <reg>	Decrement simple registre
INX <dr>	Increment double registre
DCX <dr>	Decrement double registre

. Chargements et rangements

MOV <r1>,<r2>	Transfert registres <r2> --> <r1>
LDAX B	Octet pointe par BC --> A
LDAX D	Octet pointe par DE --> A
STAX B	A --> octet pointe par BC
STAX D	A --> octet pointe par DE
LHLD adr	Mot pointe par adr --> HL
SHLD adr	HL --> mot pointe par adr
LDA adr	Octet pointe par adr --> A
STA adr	A --> octet pointe par adr
POP <pr>	pile --> registre <pr>
PUSH <pr>	Registre <pr> --> pile
XTHL	Echange haut de pile avec HL
PCHL	HL --> pointeur programme
SPHL	HL --> pointeur de pile
XCHG	Echange DE et HL

. Entrées/sorties

IN <port>	Lecture <port> --> A
OUT <port>	Sortie A --> <port>

. Arithmétiques et logiques

ADD <reg>	Ajoute contenu de <reg> a A
ADC <reg>	Ajoute contenu de <reg>+Carry a A
SUB <reg>	Soustrait contenu de <reg> a A
SBB <reg>	Soustrait contenu de <reg>+Cy a A
ANA <reg>	ET logique entre <reg> et A --> A
XRA <reg>	OU excusif entre <reg> et A --> A
ORA <reg>	OU logique entre <reg> et A --> A
CMP <reg>	Compare <reg> et A
DAA	Ajustement decimal de A
CMA	Complement logique de A
STC	1 --> Carry
CMC	Complemente Carry
RLC	Decalage gauche de A
RRC	Decalage droit de A
RAL	Rotation gauche de A avec Carry
RAR	Rotation droite de A avec Carry
DAD <dr>	Addition (HL)+<dr> --> HL

. **Contrôles**

HLT	Arret du microprocesseur
DI	Invalidation interruption
EI	Validation interruption
NOP	instruction sans effet

Signification des erreurs ASM

D Erreur Data
E Erreur Expression (ne peut etre calculee)
L Erreur dans le Label
N Non implemente dans cette version d'ASM
O Debordement dans un calcul d'expression
P Erreur de phase
R Registre incorrect
V Valeur d'operande impropre

NO SOURCE FILE PRESENT : fichier source absent
NO DIRECTORY SPACE : catalogue plein
SOURCE FILE NAME ERROR : nom de fichier incorrect
SOURCE FILE READ ERROR : erreur de lecture dans source
OUTPUT FILE WRITE ERROR : erreur d'ecriture fichier sortie
CANNOT CLOSE FILE : fichier ne pouvant pas etre ferme

STRUCTURE DES FICHIERS CP/M

Un fichier est constitué d'enregistrements. *Un enregistrement est un bloc de 128 octets consécutifs.*

Au niveau de la disquette, *les enregistrements sont écrits sous forme de secteurs.*

Pour des raisons de performances, ces secteurs peuvent être de 128 octets (1 enregistrement), mais aussi de 256, 512 ou 1024 octets (2, 4 ou 8 enregistrements). Dans ce dernier cas, il est nécessaire, au cours des lectures/écritures disque, de faire appel à un mécanisme dit de "blocage/déblocage", chargé de convertir les enregistrements en secteurs et vice-versa.

Un fichier est défini, au moment de sa création, par une entrée dans le catalogue de la disquette, dont on retrouve les éléments dans le FCB (voir FCB en appendice). Le premier octet de cette entrée indique si le fichier est présent ou non sur la disquette.

L'espace disquette, du fait de la possibilité de pouvoir effacer des fichiers, est formé de zones disjointes, séparées par les zones des fichiers ayant été supprimés.

Lors de la création d'un fichier, le système va donc essayer de combler ces espaces inutilisés, ce qui conduit à avoir des fichiers dont les enregistrements ne sont pas forcément consécutifs. Toutefois, la zone minimum, appelée **unité d'allocation**, est formée de 8 secteurs et tout fichier, même si sa longueur est de quelques octets, occupera physiquement 8 secteurs sur la disquette. On peut ainsi définir 16 unités d'allocation dans une entrée du catalogue. Lorsque ces 16 entrées ne suffisent plus, le système crée une seconde entrée dans le catalogue, appelée **extension.** On peut donc trouver plusieurs entrées relatives à un même fichier dans le catalogue de la disquette.

Lorsqu'un fichier est appelé pour être lu ou chargé en mémoire, le catalogue est parcouru séquentiellement par le système jusqu'à trouver l'entrée correspondant à ce fichier. A partir des unités d'allocations décrites dans cette entrée, il est possible de charger les secteurs du fichier.

Pour compliquer le tout, le **BIOS** possède une **table de translation** des secteurs. Vus de l'utilisateur, les secteurs composant un fichier sont rangés séquentiellement : secteur 1, secteur 2, secteur 3... Sur une disquette 8 pouces, on en trouve 26 par piste en simple densité (IBM 3740). Physiquement sur la disquette, ils sont, en fait, dans un ordre différent : 1, 7, 13, 19, 25... Cet "entrelacement" permet d'optimiser le temps de chargement des secteurs en mémoire (en cas de traitement "unité centrale" et si les secteurs sont consécutifs sur la disquette, on risque en effet de devoir attendre un tour de piste supplémentaire pour saisir le secteur suivant).

La table de translation du **BIOS** décrit donc l'entrelacement des secteurs sur la disquette. De là, vient parfois un problème de compatibilité entre les logiciels soi-disant compatibles CP/M... Il reste à l'utilisateur de devoir modifier cette table dynamiquement, par DDT par exemple.

BASIC SOUS CP/M

L'un des interpréteurs Basic parmi les plus connus et les plus performants est le MBASIC de Microsoft.

Nous avons choisi de vous présenter un petit exemple de programme réalisé sous CP/M avec ce langage. Il s'agit d'un programme de DUMP mémoire, analogue à celui donné par la commande D de DDT.

Pour charger MBASIC, une configuration mémoire de 20K n'est pas suffisante et il est nécessaire d'utiliser MOVCPM ou de partir d'un système généré avec le maximum de mémoire disponible.

Voici la liste et le "RUN" de ce programme.

```
62k CP/M vers 2.2

A)mbasic
BASIC-80 Rev. 5.2
[CP/M Version]
Copyright 1977, 78, 79, 80 (C) by Microsoft
Created: 14-Jul-80
32838 Bytes free
Ok
```

```
load "dumpm.bas"
Ok
list
10 DEFINT A-Z
20 PRINT STRING$(25,CHR$(10))
30 PRINT TAB(10) ; "D U M P   M E M O I R E"
40 PRINT TAB(10) ; "-----------------------------" : PRINT
50 PRINT : INPUT "- Adresse debut (hexa sans H)" ; AD$
60 INPUT "- Adresse fin   (hexa sans H)" ; AF$
70 PRINT
80 AD=VAL("&H"+AD$) : AF=VAL("&H"+AF$)
90 IF AD>AF THEN PRINT "erreur..." : GOTO 50
100 ALPHA$=""
110 PRINT RIGHT$("000"+HEX$(AD),4) ; ": " ;
120 FOR N=1 TO 16
130 A=PEEK(AD) : A$=CHR$(A)
140 IF A<&H20 OR A>&H7F THEN A$="."
150 ALPHA$=ALPHA$+A$
160 PRINT RIGHT$("0"+HEX$(A),2) ; " " ;
170 AD=AD+1
180 NEXT N
190 PRINT "  " ; ALPHA$
200 IF AD<AF THEN 100 ELSE 50
Ok
```

```
run
```

D U M P M E M O I R E

```
- Adresse debut (hexa sans H)? 0
- Adresse fin   (hexa sans H)? ff

0000: C3 03 F2 01 00 C3 06 E4 B3 FC 1D FD 1C FE 3B F8   ..............;.
0010: 01 00 B3 FC 1D FD 1C FE 3B F8 01 00 B3 FC 1D FD   ........;.......
0020: 1C FE 3B F8 01 00 B3 FC 1D FD 1C FE 3B F8 01 00   ..;.........;...
0030: B3 FC 1D FD 1C FE 3B F8 C3 00 F8 00 39 00 39 00   ......;.....9.9.
0040: 39 00 39 00 39 00 39 00 39 00 39 00 39 00 39 00   9.9.9.9.9.9.9.9.
0050: 39 00 39 00 39 00 39 00 39 00 39 00 00 20 20 20   9.9.9.9.9.9..
0060: 20 20 20 20 20 20 20 20 00 00 00 3F 00 20 20 20           ...?.
0070: 20 20 20 20 20 20 20 20 00 00 00 00 00 F3 24 F3           ......$.
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00A0: 00 00 00 00 00 00 00 00 00 00 80 64 34 36 00 09   ............d46..
00B0: 10 7F 80 00 AC 51 AF 06 00 C9 01 09 00 2A B4 7B   ....Q.......*.é
00C0: 09 22 B4 7B 01 09 00 2A B6 7B 09 22 B6 7B 2A B8   .".é...*.é.".é*.
00D0: 7B 2B 22 B8 7B 21 B4 7B 7E 2C 46 D6 09 4F 78 DE   é+".é!.é~,F..Ox.
00E0: 00 47 79 C9 21 C8 7B 71 23 70 2A C8 7B 23 22 CA   .Gy.!.éqHp*.éH".
00F0: 7B 21 CF 7B 36 00 2E CC 36 01 DB F3 B1 F2 24 F3   é!.é6...6.....$.

- Adresse debut (hexa sans H)? 4b0
- Adresse fin   (hexa sans H)? 4ff

04B0: 1B 32 29 00 4E 45 58 54 20 77 69 74 68 6F 75 74   .2).NEXT without
04C0: 20 46 4F 52 00 53 79 6E 74 61 78 20 65 72 72 6F    FOR.Syntax erro
04D0: 72 00 52 45 54 55 52 4E 20 77 69 74 68 6F 75 74   r.RETURN without
04E0: 20 47 4F 53 55 42 00 4F 75 74 20 6F 66 20 44 41    GOSUB.Out of DA
04F0: 54 41 00 49 6C 6C 65 67 61 6C 20 66 75 6E 63 74   TA.Illegal funct

- Adresse debut (hexa sans H)?
```

CPM-80 ET CPM-86

Depuis peu de temps, les micro-ordinateurs 16 bits commencent à faire leur apparition sur le marché et la plupart d'entre eux comportent un microprocesseur Intel 8086 (architecture matérielle et logicielle 16 bits) ou 8088 (architecture logicielle 16 bits et architecture matérielle 8 bits).

Ces microprocesseurs étant dans la lignée du 8080, Digital Research a donc développé un système d'exploitation spécifique appelé CP/M-86, le précédent CP/M prenant alors le nom de "CP/M-80".

Pour ceux qui connaissent déjà CP/M, ce nouveau système d'exploitation sera très facile à assimiler. Voici la liste des quelques remarques que nous avons notées.

. CP/M-86 est capable de gérer jusqu'à 1 méga-octet de mémoire (par segments de 64K octets) et jusqu'à 16 unités de disques (ou disquettes) de 8 Moctets chacun.

. La structure des fichiers est, en principe, compatible avec celle du CP/M-80.

. Les commandes CP/M-80 sont conservées mises à part ces quelques différences :

- LOAD devient GENCMD
- ASM devient ASM86
- DDT devient DDT86
- SYSGEN devient LDCOPY

. Les types de fichier ".COM" et ".HEX" deviennent respectivement ".CMD" et ".H86".

. Les "mnémoniques" des instructions de l'assembleur ne sont évidemment plus celles du 8080 mais celles du 8086. Il s'en suit que le code machine n'est plus le même.

. DDT86 tient compte de la capacité mémoire et des instructions apportées par ce nouveau microprocesseur.

. L'utilitaire LDCOPY (LoaDer COPY) permet de recopier le chargeur sur les deux premières pistes de la disquette. A ce sujet, il faut signaler que le mécanisme de chargement est légèrement différent - mais bien plus pratique pour personnaliser le système. A l'initialisation, lorsque le chargeur est transféré en mémoire par la ROM de l'ordinateur, celui-ci a la faculté de consulter le catalogue de la disquette et de charger le fichier nommé "CPM.SYS". Ce fichier constitue le système d'exploitation et est formé du CCP, du BDOS et du BIOS fournis sur la disquette Digital Research sous la forme de fichiers ".H86". De même, le chargeur est composé d'un "mini" CP/M et d'un "mini" BIOS. Toutes les facilités sont données à l'utilisateur pour que celui-ci puisse générer facilement un chargeur (LOADER.H86) et un CP/M (CPM.SYS) adaptés à son système, ce qui n'était pas tout à fait le cas avec CP/M-80. De plus, la limite imposée par les deux premières pistes de la disquette pour contenir le système (limitant donc physiquement la taille du BIOS) n'existe plus, puisque celui-ci est maintenant considéré comme un fichier.

. CCP est toujours résident en mémoire (n'est plus rechargé par le démarrage à chaud).

. La commande MOVCPM n'existe plus (les programmes sont relogeables en 8086).

. La commande SAVE n'existe plus (DDT86 offre une commande équivalente).

. Le nouvel utilitaire LMCMD permet de convertir des fichiers Intel (issus de la commande LOCATE) en fichiers ".CMD".

. Les appels BDOS s'effectuent maintenant par l'intermédiaire d'une interruption. Ceux-ci ont été enrichis de 10 entrées supplémentaires.

Les documents de base de CP/M-86 sont les mêmes que ceux de CP/M-80 délivrés par Digital Research, mis à part :

- User's guide ASM86 (remplace ASM)

- User's guide DDT86 (remplace DDT)

- CP/M-86 System Reference guide (remplace Interface guide et Altération guide).

CODE ASCII

POIDS									
forts / faibles		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	VS	/	?	O	_	o	DEL

BIBLIOGRAPHIE

Documents d'origine Digital Research (fournis avec CP/M) :

- CP/M 2.2 Interface Guide
- CP/M 2.2 Alteration Guide
- An Introduction to CP/M features and facilities
- ED : A context Editor for CP/M (User's manual)
- CP/M Assembler (User's manual)
- CP/M Dynamic Debugging Tool (User's manual)

Autres documents utiles :

- CP/M Primer, par S. Murtha et M. Waite, édité par Howard W. Sams & Co, Indianapolis, IN
- Osborne CP/M User Guide, par Thom Hogan, édité par Osborne/ McGraw-Hill, Berkeley CA
- Guide du CP/M (avec MP/M), par Rodnay Zaks, édité par Sybex, Paris
- How to get started with CP/M, par Carl Townsend, édité par Dilithium Press (éditions du PSI)
- Dr Dobb's Journal numéro 41 (jan 80) - numéro spécial consacré à CP/M, édité par People's computer, Menlo Park CA

RESUME DES COMMANDES CP/M

Définitions

<ref explicite> désigne un nom de fichier suivi éventuellement d'un type. Le nom de la disquette, s'il est présent, précède ce nom (A à P).

<nom disquette>:<nom fichier>.<type>

avec <nom de fichier> sur 1 à 8 caractères et <type> sur 1 à 3 caractères.

<ref ambigüe> est identique à <ref explicite> sauf que le nom et/ou le type peuvent contenir les caractères "*" ou "?".

Commandes résidentes

DIR	d: <ref ambigüe> <ref explicite>	Listage du catalogue
ERA	<ref explicite> <ref ambigüe>	Destruction d'un fichier
REN	<nouvelle ref explicite>=<ancienne ref explicite>	Changement de nom d'un fichier
SAVE	p <ref explicite>	Sauvegarde de p pages mémoire sur disque
d:		Affectation disquette implicite
TYPE	<ref explicite>	Affichage d'un fichier ASCII
USER	z	Changement zone (z) utilisateur

Commandes transitoires

ASM	<ref explicite sans type>.**SOL**	Assemblage d'un fichier source
DDT	<ref explicite>	Appel utilitaire de mise au point
DUMP	<ref explicite>	Listage d'un fichier ASCII
ED	<ref explicite>	Appel de l'éditeur
LOAD	<ref explicite sans type>	Génération d'un fichier .COM à partir d'un fichier .HEX (implicite)
MOVCPM	<val> <val> *	Création d'une configuration CP/M chargée (*) ou non en zone TPA pour sauvegarde ou modification
PIP	<ref explicite>	Appel de la commande de transfert fichiers/périphériques
SUBMIT	<ref explicite> <liste de paramètres>	exécution d'un fichier de commande
STAT	<d:> **DEV:** **VAL:**	Listage : - des états disquettes - des assignements périphériques - des assignements valides

DSK:	- des caractéristiques disquette
USR:	- des zones utilisateurs
<ref explicite> ou <ref ambigüe>	- de la taille d'un fichier
<ref explicite ou ambigüe> $S	Taille réelle des fichiers
<ref explicite ou ambigüe> $<attribut>	Changement d'attribut d'un fichier : R/O : protection écriture R/W : validation écriture SYS : invisibilité DIR : visibilité
<d:>=R/O	Protection temporaire disquette
<organe logique>=<organe physique>	Changement d'affectation d'un organe périphérique
SYSGEN	Commande de génération de système
XSUB	Extension de la commande SUBMIT

Paramètres et mots-clés employés par PIP

B	Lecture par bloc
Dn	Troncature de lignes
E	Echo sur la console
F	Suppression des sauts de page (OCH)
Gn	Fichier source dans la zone utilisateur n
H	Contrôle du format hexa HEX
I	Idem H et supprime les EOF
L	Conversion en minuscules
N	Numérotation lignes
O	Ignore EOF (fichier objet)
Pn	Saut de page toutes les n lignes (60 si n absent ou = 1)
Q<chaîne>↑Z	Quitte le transfert après chaîne
R	Lecture fichier invisible (SYS)
S<chaîne>↑Z	Démarre le transfert après chaîne
Tn	Expansion des tabulations a n col
U	Conversion en majuscules
V	Force vérification transfert disque
W	Fichier destination remplace si R/O
Z	Force parité à zéro en entrée
EOF:	Génération d'un CTRL Z (fin de fichier)
INP:	Organe d'entrée (modif PIP)
NUL:	Génération amorce (40 car nuls)
OUT:	Organe de sortie (modif PIP)
PRN:	Idem LST: mais avec présentation

Commandes DDT

A<adr>	Assemblage à partir de <adr>
D<add>,<adf>	Dump de <add> à <adf>
F<add>,<adf>,<val>	Remplissage d'une zone mémoire, de <add> à <adf>, avec la valeur <val>
G<adr>,<bk1>,<bk2>	Exécution programme à partir de <adr> avec adresses d'arrêt <bk1> et <bk2>
H<x>,<y>	Calcul de x+y et x-y
I<nom fichier>	Définit un nom de fichier (voir R)
L<add>,<adf>	Désassemblage de <add> à <adf>

M<add>,<adf>,<dest>	Déplacement zone mémoire <add> à <adf> à partir de <dest>
R<deplact>	Lecture du fichier défini par I avec déplacmemnt éventuel
S<adr>	Substitution mémoire en <adr>
Tn	Trace de n instructions
Un	Exécute n instructions sans trace
X<r>	Examen registres

Commandes ED

nA	Appel n lignes dans le buffer
B	Pointeur en début de buffer
-B	Pointeur en fin de buffer
nC ou **-nC**	Déplacement de n carac. (AV ou AR)
nD ou **-nD**	Suppression de n carac. (AV ou AR)
E	Fin d'édition avec mise à jour
nF<chaîne>	Recherche n fois <chaîne> dans le buffer
H	Exécute E puis retourne dans ED
I<texte>↑Z	Insertion <texte>
nK ou **-nK**	Suppression de n lignes (AV ou AR)
nL ou **-nL**	Déplacement de n lignes (AV ou AR)
nM<commande>	Exécute n fois <commande>
nN<chaîne>	Idem F mais dans tout le fichier
O	Ré-initialise ED (sans mise à jour)
nP	Listage par page
Q	Quitte ED (sans mise à jour)
R	Insertion de X$$$$$$$$.LIB
R<fich>	Insertion du fichier <fich>.LIB
nS<ch1>↑**Z**<ch2>	Remplace n fois la chaîne <ch1> par la chaîne <ch2>
nT	Affiche n lignes
U	Conversion minuscules en majuscules
-V ou **V**	Supprime ou rétablit les numéros de lignes du fichier
nW	Ecrit n lignes dans fichier de sortie (depuis début buffer)
nX	Ecrit n lignes dans X$$$$$$$.LIB
nZ	Arrêt opération pendant un temps dépendant de n

Achevé d'imprimer en septembre 1983
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : septembre 1983

N° d'impression : 308051
N° d'édition : 86595-51-3
ISBN : 2-86595-051-4

GUIDE PRATIQUE

*

CP/M PAS A PAS

Cet ouvrage s'adresse au lecteur désireux de pratiquer le système d'exploitation de disquettes CP/M. Dans ce but, il décrit de manière pédagogique et avec de nombreux exemples à l'appui, toutes les commandes de ce moniteur et de ses utilitaires. Il est hautement souhaitable que le lecteur possède un ordinateur muni de CP/M, et sur lequel il puisse pratiquer. Les connaissances requises sont modestes : avoir quelques notions d'informatique générale et connaître si possible, le langage assembleur (mais ce n'est pas impératif).

Editions du P.S.I.
Boîte Postale 86
F - 77402 Lagny/Marne Cedex

Graphisme : Françoise Guillot

ISBN : 2 86595 051 4 - Imprimé en France

Document numérisé
avec amour par :
AMSTRAD
CPC
MÉMOIRE ÉCRITE
ACME
https://acpc.me/